Large Margin Transformation Learning

Andrew G. Howard

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2009

ABSTRACT

Large Margin Transformation Learning

Andrew G. Howard

With the current explosion of data coming from many scientific fields and industry, machine learning algorithms are more important than ever to help make sense of this data in an automated manner. Support vector machine (SVMs) have been a very successful learning algorithm for many applied settings. However, the support vector machine only finds linear classifiers so data often needs to be preprocessed with appropriately chosen nonlinear mappings in order to find a model with good predictive properties. These mappings can either take the form of an explicit transformation or be defined implicitly with a kernel function.

Automatically choosing these mappings has been studied under the name of kernel learning. These methods typically optimize a cost function to find a kernel made up of a combination of base kernels thus implicitly learning mappings. This dissertation investigates methods for choosing explicit transformations automatically. This setting differs from the kernel learning framework by learning a combination of base transformations rather than base kernels. This allows prior knowledge to be exploited in the functional form of the transformations which may not be easily encoded as kernels such as when learning monotonic transformations. Additionally, kernel based SVMs are often hard to interpret because they lead to complex decision boundaries which are only linear in the implicitly defined space. However, by working with explicit mappings, models with an intuitive meaning can be learned. The learned transformations can be visualized to lend insight into the problem, and the hyperplane weights indicate the importance of transformed features.

The two basic models that will be studied are the a mixture of transformations $\Phi(x) = \sum_i m_i \phi_i(x)$ and the matrix mixture of transformations which defines kernels of the form $k(x, x') = \sum_{i,j} m_i m_j \phi_i(x)^T \phi_j(x')$. The matrix mixture reduces to mixture of transfor-

mation learning when M is rank 1 and mixture of kernel learning when M is a diagonal matrix. First, greedy algorithms are proposed to simultaneously learn a mixture of transformations and a large margin hyperplane classifier. Then, a convex semidefinite algorithm is derived to find a matrix mixture of transformations and large margin hyperplane. More efficient algorithms based on the extragradient method are introduced to solve larger problems and extend the basic framework to a multitask setting. Another cost function based on kernel alignment is explored to learn matrix mixture of transformations. Maximizing the alignment with a cardinality constraint on the mixture weights gives rise to approximation algorithms with constant factor approximations similar to the Max-Cut problem. These methods are then applied to the task of learning monotonic transformations which are built from a mixture of truncated ramp functions. Experimental results for synthetic data, image histogram classification, text classification and gender recognition demonstrate the utility of these learned transformations.

Contents

List of Figures

List of Tables

This dissertation was only possible with the help and support of my family, friends, colleagues and mentors. Thank you.

Chapter 1

Introduction

1.1 Preliminaries

In the past decade, large margin classifiers based on the support vector machine formulation have been very successful in applied domains [CV95, SS02]. A large part of this success can be attributed to the use of explicit nonlinear transformations or implicit mappings defined by reproducing kernels. These mappings are used so that the resulting linear decision boundary in the new space is nonlinear in the original space thus extending the flexibility of a linear classifier. An open question and subject of much research is how to pick an appropriate transformation or kernel. Although there are various heuristics and rules of thumb that are used in practice, approaches that learn the kernel from data are receiving much attention [BLJ04, SRSS06, RBCG08, CSTK02, OSW05, BH03, CKS02, WB98, WR95, TAA03, LJN06].

This dissertation investigates a new data driven method to learn transformations that extends the previous kernel learning paradigm. We start with a mixture of transformations and formulate a joint optimization of the mixing weights and a maximum margin hyperplane. This nonconvex optimization is relaxed to yield a convex semidefinite program (SDP) which learns a matrix mixture of transformations, a formulation that can be seen as a generalization of the multiple kernel learning framework [LCB$^+$04]. The large margin

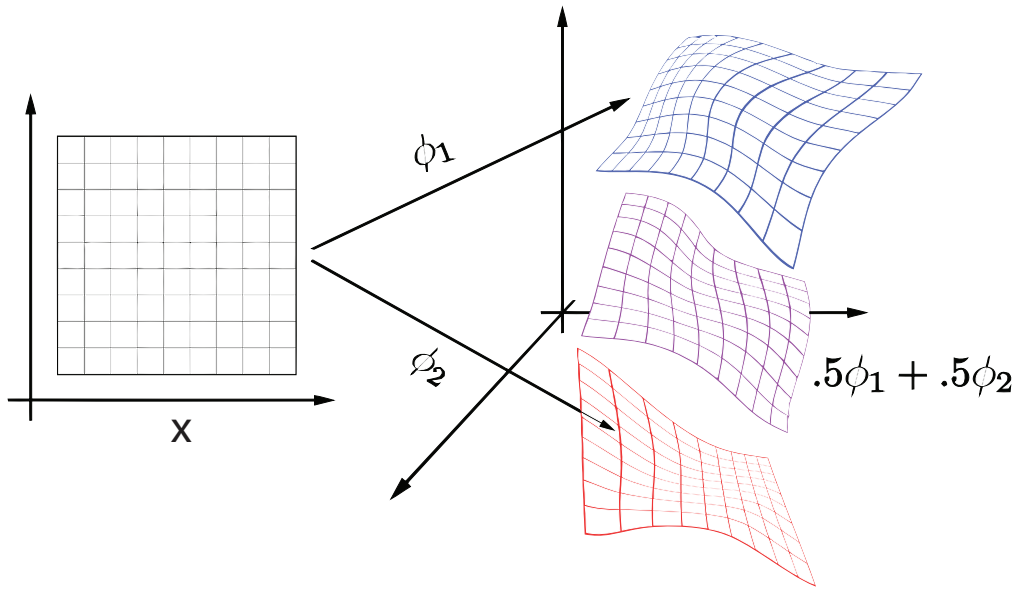


Figure 1.1: The input space (left) is mapped to a transformed space by two transformations $\phi_1(x)$, $\phi_2(x)$ and their mixture $.5\phi_1(x) + .5\phi_2(x)$ (right).

mixture of transformations formulation was initially proposed to learn monotonic transformations from a mixture of truncated ramp functions [HJ08b] and extended to a general setting in [HJ08a].

Previous kernel learning methods learned a hyperplane in conjunction with a kernel made up of a mixture of base kernels, $k(\vec{x}, \vec{x}') = \sum_i m_i k_i(\vec{x}, \vec{x}')$. Rather than learning a mixture of kernels, we will investigate mixtures of transformations $\Phi(\vec{x}) = \sum_i m_i \phi_i(\vec{x})$ as depicted in Figure 1.1. This leads to learned kernels of the form: $k(\vec{x}, \vec{x}') = \sum_{i,j} m_i m_j \phi_i(\vec{x}) \phi_j(\vec{x}')$. We also explore the matrix mixture of transformations: $k(\vec{x}, \vec{x}') = \sum_{i,j} M_{i,j} \phi_i(\vec{x}) \phi_j(\vec{x}')$ which subsumes the previous two formulations. The mixture of kernels can be replicated with a diagonal M and the mixture of transformations can be replicated when M is rank one.

Chapter 1 presents an introduction to the large margin transformation learning problem and reviews some important basics of kernel methods, optimization and previous kernel learning algorithms. Chapter 2 formally introduces the large margin mixture of transfor-

mation learning problem and proposes greedy local algorithms for solving it. In Chapter 3, a convex relaxation is derived and cast as a semidefinite program to find approximate global solutions. This relaxed optimization defines the matrix mixture of transformations. Chapter 4 applies the extragradient algorithm to the learning task in order to handle larger datasets and as a stepping stone to various extensions. Chapter 5 generalizes the mixture of transformations to multitask learning where one transformation is used in multiple related tasks. Chapter 6 deals with the specific case of learning monotonic transformations. Chapter 7 looks at learning mixtures of transformations via kernel alignment and develops connections between sparse transformation learning via alignment and constant factor approximable NP Hard combinatorial problems. Chapter 8 demonstrates the usefulness of the proposed methods in experiments. We conclude in Chapter 9 with a discussion and future work.

1.2 Kernel Methods

Kernel methods typically start with a linear algorithm and replace the inner products with inner products between (implicitly) mapped points through the use of a kernel function $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$. Kernel functions allow these linear algorithms to find nonlinear decision surfaces and to extend algorithms to cases where the data is made up of more complicated objects than vectors. Some basic examples of kernels are the radial basis function (RBF):

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \tag{1.1}$$

and the polynomial kernel:

$$k(x, x') = (\langle x, x' \rangle + 1)^2. \tag{1.2}$$

These kernels are defined on vectors in $\Re^n$ but more complicated kernels leveraging the structure on the underlying objects have been explored. An example of a general method to derive these more complicated kernels is the probability product kernel (PPK) [JKH04]. The PPK is a kernel defined on probability distributions. These probability distributions capture the structure and subtleties of the data and are used in the following kernel:

$$k(p, p') = \int_x p(x)^\rho p'(x)^\rho dx \tag{1.3}$$

where $p(x)$ and $p'(x)$ are probability distributions defined by data points x and x'. For the case of $\rho = 1/2$, this is simply the Bhattacharyya affinity between distributions [Bha43]. The PPK also reduces to the RBF kernel when $p(x)$ and $p'(x)$ are Gaussian distributions with the same isotropic covariance and $\rho = 1$. Many other kernels exist to take advantage of the unique structure of data such as rational kernels for sequence data [CHM04], Boolean kernels for learning conjunctions of Boolean variables [KS05] and diffusion kernels for learning data on a graph [KL02].

Because we are going to be focusing on learning explicit mappings, it is important to characterize what types of function $k(x, x')$ give rise to implicit mappings. The Mercer condition gives necessary and sufficient conditions.

Theorem 1 *A symmetric and real valued function $k(x, x') \in L_\infty$ can be expressed as $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$ if and only if $k(x, x')$ satisfies*

$$\int k(x, x') g(x) g(x') dx dx' \geq 0 \tag{1.4}$$

for all $g(x) \in L_2$.

This follows from Mercer's theorem which defines the eigenfunction expansion of a kernel $k(x, x') = \sum_j \lambda_j \Psi(x) \Psi_j(x')$ for $k(x, x')$ satisfying theorem 1 where $\Psi_j(x)$ is the normalized orthogonal eigenfunctions associated with the eigenvalue λ_j for the integral operator T_k defined by $k(x, x')$:

$$(T_k f)(x) := \int k(x, x') f(x') dx'. \tag{1.5}$$

This defines the mapping defined by $k(x, x')$ as a concatenation of these basis functions $\Phi(x) = \left(\sqrt{\lambda_j} \Psi_j(x)\right)_{j=1,\ldots,N}$.

The kernel method that is the main topic of this document is the support vector machine algorithm for classification. The support vector machine produces a large margin linear classifier over inputs $\vec{x} \in \Re^D$ which makes binary predictions $y \in \{-1, 1\}$. The classification rule, $f(x) = \text{sign}(\vec{w}^T \vec{x} + b)$, is defined by an unknown hyperplane with parameters $\vec{w}$, b. This hyperplane can be learned from training data $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ by solving a very well studied quadratic program:

$$\begin{aligned} \min_{\vec{w},\vec{\xi},b} \quad & \|\vec{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i\left(\vec{w}^T\vec{x}_i + b\right) \geq 1 - \xi_i \,\forall i \\ & \xi_i \geq 0 \,\forall i \end{aligned} \tag{1.6}$$

where $\vec{\xi}$ are the standard SVM slack variables.

The usefulness of support vector machines often hinges upon finding, either manually or automatically, good nonlinear transformations $\Phi(\vec{x})$ or equivalently, kernel functions $k(\vec{x},\vec{x}') = \Phi(\vec{x})^T\Phi(\vec{x}')$. In order to use these kernel functions with the SVM, an alternative quadratic program is derived from the dual of the SVM formulation which is expressed using only inner products between pairs of data points. This yields the following quadratic program only expressed in terms of kernel functions:

$$\begin{aligned} \min_{\vec{\alpha}} \quad & -2\sum_i \alpha_i + \sum_{i,j}\alpha_i\alpha_j y_i y_j k\left(\vec{x}_i,\vec{x}_j\right) \\ & 0 \leq \alpha_i \leq C \,\forall i\,, \sum_i \alpha_i y_i = 0 \end{aligned} \tag{1.7}$$

where α_i is the i'th dual variable corresponding to the classification constraint for x_i.

1.3 Semidefinite Programming and Optimization

In this dissertation, we will be making extensive use of convex programming. One of the main forms of interest will be semidefinite programs (SDP). There are two basic forms that these SDPs can take. The first is the inequality form defined by optimizing over a vector with a linear cost function, a linear matrix inequality (LMI) and a linear equality constraint:

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{subject to:} \quad & F_0 + x_1F_1 + \ldots + x_mF_m \preceq 0 \\ & Ax = b \end{aligned} \tag{1.8}$$

Another equivalent formulation is the so-called standard form which is defined as an optimization over a positive semidefinite matrix with a linear cost function, linear equality constraint, and positive semidefinite constraint:

$$\begin{aligned} \max_{X} \quad & \text{trace}(CX) \\ \text{subject to} \quad & \text{trace}(A_i X) = b_i \\ & X \succeq 0. \end{aligned} \tag{1.9}$$

The inequality and standard forms are Lagrange dual pairs and therefore can represent the same class of problems. SDPs are of great interest because they generalize linear, and quadratic programs as well as define many new practical problems. Additionally, they can be solved efficiently with interior point methods.

Another useful concept which will be used in many of the subsequent derivations and relaxations is Lagrange duality. Given an arbitrary optimization problem which will be called the primal:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0 \\ & g_j(x) \leq 0, \end{aligned} \tag{1.10}$$

the Lagrangian function is defined as:

$$L(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \mu^T g(x). \tag{1.11}$$

The Lagrangian dual function is then defined as:

$$q(\lambda, \mu) = \inf_{x \in \Re^n} L(x, \lambda, \mu) \tag{1.12}$$

with $\lambda \in \Re^m, \mu \in \Re^r$ and $\mu \geq 0$.

This dual function is concave in λ and μ and has a convex domain. For all feasible settings of λ, μ, the Lagrange dual function gives a lower bound on the optimum of the

primal $q(\lambda, \mu) \leq p^*$ where $p* = \min_x f(x)$. Because $q(\lambda, \mu)$ is concave with a convex domain, the supremum can be found in polynomial time $q^* = \sup_{\lambda,\mu} q(\lambda, \mu)$. This best lower bound q^* can be used to find efficient approximations which is known as weak duality. The difference between the optimal solution and the lower bound is known as the dual gap $p^* - q^*$. Optimizing the dual function is also useful when an exact solution is sought. A sufficient condition for zero dual gap is known as Slater's condition which states that if the original problem is convex and strictly feasible then $p^* = q^*$. Optimizing the Lagrange dual function in both the exact and the approximate case will be useful in many of the derivations that follow.

Another useful technique for converting convex optimizations into SDPs is the Schur complement lemma. It gives a recipe for converting certain quadratic forms into linear matrix inequalities.

Theorem 2 *Let* $A \succ 0$ *and* $X = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$. *Then*

$$X \succ 0 \iff S = C - B^T A^{-1} B \succ 0$$

where S *is called the Schur complement of* X.

1.4 Related Kernel Learning Techniques

We now look at previously proposed kernel learning algorithms. The simplest way to choose a kernel is to cross validate over a finite fixed set of base kernel functions $\{k_1, \ldots, k_M\}$ that are manually specified. Equivalently, one may specify a set of Gram matrices $K_1, \ldots, K_M$ where an individual Gram matrix $K \in \Re^{N \times N}$ is a positive semidefinite matrix defined elementwise as $K(i, j) = k(\vec{x}_i, \vec{x}_j)$. The SVM problem is then solved for each kernel individually, and the kernel with the best cross validation score is used. This method suffers from the fact that it can only explore a small set of kernels and requires the SVM learning algorithm to be run many times. This has led to many proposed strategies for automatically searching a larger space of kernels.

A general convex optimization framework for learning the kernel has been proposed as

the following min max problem [LCB+04]:

$$\min_{K \in \mathcal{K}} \max_{\vec{\alpha}} 2\vec{\alpha}^T \vec{1} - \vec{\alpha}^T (Y K_{tr} Y) \vec{\alpha} : \vec{0} \leq \vec{\alpha} \leq C\vec{1}, \vec{\alpha}^T \vec{y} = 0 \tag{1.13}$$

where $\mathcal{K}$ is a convex set of positive semidefinite kernel matrices over training and testing data, K_{tr} is the kernel matrix involving only the training data, $\vec{1}$ is the vector of ones, $\vec{0}$ is a vector of zeros, X is a matrix with $\vec{x}_i$ in the ith column, Y is a matrix with the training labels on the diagonal with zeros elsewhere and $\vec{y}$ is a vector of training labels. The problem as specified solves the transduction problem [Vap98] by finding kernels over both training and testing data, but the optimization can also be restricted to Gram matrices of only training data to solve the inductive learning problem.

Different learning algorithms are derived depending on how the set $\mathcal{K}$ is specified. Typically, $\mathcal{K}$ is chosen to be a linear combination of base kernels. In order to ensure that a valid kernel is learnt, a positive semidefinite constraint is applied to the kernel matrix:

$$K = \sum_{i=1}^{M} m_i K_i,\ K \succeq 0\ \forall i. \tag{1.14}$$

This requires a semidefinite program (SDP) to learn the kernel:

$$\begin{aligned} \min_{\vec{m},t,\lambda,\vec{\nu},\vec{\delta}} \quad & t \\ \text{subject to} \quad & \text{trace}\left(\sum_{j=1}^{J} m_j K_j\right) = c \\ & \sum_{j=1}^{J} m_j K_j \succeq 0 \\ & \begin{pmatrix} Y\left(\sum_{j=1}^{J} m_j K_j\right) Y & \vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \\ (\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y})^T & t - 2C\vec{\delta}^T \vec{1} \end{pmatrix} \succeq 0 \\ & \vec{\nu} \geq \vec{0}, \vec{\delta} \geq \vec{0}. \end{aligned} \tag{1.15}$$

By using non-negativity constraints on the mixing weights m_i, semidefiniteness of the learned kernel is enforced if each base kernel is a valid positive semidefinite matrix:

$$K = \sum_{i=1}^{M} m_i K_i,\ m_i \geq 0\ \forall i. \tag{1.16}$$

Kernels of this form are learned with a quadratically constrained quadratic program (QCQP):

$$\begin{aligned} \max_{\vec{\alpha},t} \quad & 2\vec{\alpha}^T\vec{1} - ct \\ \text{subject to} \quad & t \geq \vec{\alpha}^T Y K_{j,tr} Y \vec{\alpha},\ j = 1,\ldots,J \\ & \vec{\alpha}^T\vec{y} = 0 \\ & C\vec{1} \geq \vec{\alpha} \geq \vec{0}. \end{aligned} \tag{1.17}$$

A large body of work has built on this framework mainly focusing on algorithmic improvements for finding conic combinations of kernels as in (1.16) [BLJ04, SRSS06, RBCG08].

In the original multiple kernel learning paper [LCB+04] a bound on the estimation error was given for simultaneously learning the kernel and the maximum margin hyperplane. The bound was in the form of $\sqrt{\tilde{O}(\frac{J}{\gamma^2})/N}$ where J is the number of kernels, γ is the margin, N is the number of training examples and $\tilde{O}()$ notation hides logarithmic factors as well as some terms involving the sample size and the failure probability. This bound was derived based on Rademacher averages but unfortunately it depends multiplicatively between the margin term $\frac{1}{\gamma^2}$ and the number of kernels. Srebro and Ben-David [SBD06] derived another bound based on covering numbers and the pseudo-dimension that was of the form $\sqrt{\tilde{O}(d_p + \frac{1}{\gamma^2})/N}$ where $d_p$is the pseudo-dimension. The pseudo-dimension is bounded by the number of base kernels J which yields a bound that is additive between the number of kernels and the margin term. They also proved that the bound proposed in [BH03] is vacuous.

The first algorithm proposed for speeding up the conic combinations of kernels were based on sequential minimal optimization (SMO) [BLJ04]. First, a new formulation called the support kernel machine was introduced:

$$\begin{aligned} \min_{\vec{w},b,\vec{\xi}} \quad & \frac{1}{2}\left(\sum_{j=1}^{J} d_j \|w_j\|_2\right)^2 + C\sum_{i=1}^{N}\xi i \\ \text{subject to} \quad & y_i\left(\vec{w}^T\vec{x}_i + b\right) \geq 1 - \xi_i\ \forall i \\ & \xi_i \geq 0\ \forall i. \end{aligned} \tag{1.18}$$

The dual of this formulation can be shown to be equivalent to the QCQP in 1.17. This formulation gives an intuition on why kernel learning often gives sparse solutions. The cost

function involves minimizing the square of a blockwise l_1 norm which sparsifies the blocks and equivalently the kernel weights. Another equivalent optimization is then proposed to derive the SMO optimization:

$$\begin{aligned} \min_{\vec{\alpha}} \quad & \max_j \{\frac{1}{2}\vec{\alpha}^T Y K_{j,tr} Y \vec{\alpha} - \vec{\alpha}^T \vec{1}\} \\ \text{subject to} \quad & \vec{\alpha}^T \vec{y} = 0 \\ & C\vec{1} \geq \vec{\alpha} \geq \vec{0}. \end{aligned} \tag{1.19}$$

SMO cannot be applied directly to this formulation because is it a nonsmooth optimization due to the maximization over j. This requires a smoothing technique called Moreau-Yosida regularization which is used in the optimization literature for nonsmooth problems.

The next algorithmic improvement to multikernel learning was based on semi-infinite linear program (SILP) [SRSS06]. A SILP was derived that yielded an equivalent solution to 1.17:

$$\max_{t,\vec{m}} \quad t \tag{1.20}$$

$$\begin{aligned} \text{subject to} \quad & \sum_j m_j \frac{1}{2}\vec{\alpha}^T Y K_{j,tr} Y \vec{\alpha} - \vec{\alpha}^T \vec{1} \geq t \\ & \vec{0} \leq \vec{m}, \sum_j m_j = 1 \\ & \forall\ \vec{\alpha} \text{ with } \vec{0} \leq \vec{\alpha} \leq C\vec{1}, \vec{y}^T \vec{\alpha} = 0. \end{aligned} \tag{1.21}$$

This is a SILP because there is an infinite number of linear inequalities based on every possible setting of $\vec{\alpha}$. The authors propose to solve this SILP with column generation which iteratively adds the most violated constraint. Finding the most violated constraint reduces to solving an SVM with a fixed kernel that allows for an efficient reuse of existing fast solvers for the SVM problem.

A gradient based method called simpleMKL [RBCG08] was proposed that allowed for a simple implementation and was demonstrated to solve the multiple kernel learning problem faster than the SILP implementation. The authors proposed optimizing the following

equivalent formulation via a reduced gradient algorithm:

$$\min_{\vec{m}} J(\vec{m}) \qquad s.t.\ \vec{0} \leq \vec{m}, \sum_j m_j = 1, \tag{1.22}$$

$$J(\vec{m}) = \begin{cases} \min_{\vec{w},b,\vec{\xi}} & \frac{1}{2}\sum_j \frac{1}{m_j}\|\vec{w_j}\|_2^2 + C\sum_i \xi_i \\ \text{s.t.} & y_i(\sum_j \vec{w_j}^T x_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \forall i. \end{cases} \tag{1.23}$$

The gradient of $J(\vec{m})$ can be expressed in terms of the dual SVM solution for the current kernel setting:

$$\frac{\partial J}{\partial m_k} = -\frac{1}{2}\sum_{i,j} \alpha_i^\star \alpha_j^\star y_i y_j K_m(x_i, x_j) \tag{1.24}$$

where $\alpha_i^\star$ is the optimal dual variable for the current kernel combination. Similar to the SILP program formulation, this method can reuse efficient SVM solvers. At the most recent kernel learning workshop, a method improving on the gradient based algorithm by incorporating second order information was shown to be more efficient than the first order gradient method [CR08].

Another method related to learning a conic combination of kernels is to learn sequence kernels based on weighted transducers [CMR08]. The method basically learns weights for transducers where the squared weights can be seen as kernel weights in the MKL framework and the base kernels are simply the inner product between the counts of the number of times a string (for example a bigram) appears as a substring in an example string. This is exactly the multikernel framework, but because each kernel's features can be calculated directly rather than implicitly, there is a more efficient QP that can be solved. This is quite similar to the original QCQP originally derived in 1.17 for learning conic combinations of kernels.

$$\begin{aligned} \max_{\vec{\alpha},t} \quad & 2\vec{\alpha}^T\vec{1} - ct^2 \\ \text{subject to} \quad & -t \leq \vec{\alpha}^T \frac{Y^T X_j}{\|X_j\|} \leq t,\ j = 1, \ldots, J \\ & \vec{\alpha}^T \vec{y} = 0 \\ & C\vec{1} \geq \vec{\alpha} \geq \vec{0} \end{aligned} \tag{1.25}$$

where $K_j = X_j^T X_j$. This QP could be used for solving the original MKL problem if each kernel were decomposed via Cholesky or eigenvalue decomposition or if the kernel's feature

vector could be computed directly.

Another interesting method suggested at the kernel learning workshop was to constrain the l_2 norm of the mixing weights instead of the l_1 norm as in the standard MKL framework [KBLS08]. The formulation was shown to be robust against noisy and redundant features. This yielded a similar semi-infinite quadratic program which can be solved with column generation and standard SVM solvers which is exactly the same as 1.20 except the l_1 constraint $\sum_j m_j \leq 1$ is replaced with the l_2 constraint $\sum_j m_j^2 \leq 1$.

Other methods for kernel learning include the kernel alignment method [CSTK02]. Contrary to the previous method, which learns the kernel and the classifier simultaneously, kernel alignment is run prior to the training algorithm. Kernel alignment is often used to maximally align a kernel with an idealized one based on the labels. This is measured by its alignment score:

$$A(K, yy^T) = \frac{\left\langle K, \vec{y}\vec{y}^T \right\rangle_F}{\sqrt{\langle K, K \rangle_F \left\langle \vec{y}\vec{y}^T, \vec{y}\vec{y}^T \right\rangle_F}} = \frac{\left\langle K, \vec{y}\vec{y}^T \right\rangle_F}{N\sqrt{\langle K, K \rangle_F}}. \tag{1.26}$$

The alignment can be optimized with convex programs similar to the previous technique:

$$\begin{aligned} \max_K \quad & \left\langle K, \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \|K\|_F = 1 \\ & K \in \mathcal{K}. \end{aligned} \tag{1.27}$$

Another interesting theme in kernel selection is to find an optimal kernel as a small combination of kernels chosen from an infinite set of base kernels. It was shown that a representer theorem exists for learning the kernel, meaning that only a finite subset of kernels form the optimal solution when learning from an infinite set of kernels [AMP05]. Initially, a greedy algorithm was defined to find an approximately optimal solution to the problem. Subsequently, a difference of convex functions (DC) programming solution was proposed to solve the problem [AHMP06].

Another recent kernel learning approach was proposed via so-called hyperkernels [OSW05]. This method defines a reproducing kernel Hilbert space on the kernels themselves. Within this framework, a representer theorem can be derived which leads to an estimation problem with a finite number of terms. Semidefinite programs are formulated to find the best

kernel in the space of hyper reproducing kernels but in practice this is computationally cumbersome. A specific class of hyperkernels called Gaussian and Wishart hyperkernels was proposed in [KJ07]. In the recent kernel learning workshop hyperkernels were used for kernel density estimation [GVG08] and for distance learning in a K nearest neighbor classifier [OG08].

As kernel learning is a currently popular subject of exploration, there are a number of other strategies for learning optimal kernels. In [BH03] gradient descent is used to optimize a bound on the Rademacher complexity of the kernel. [CKS02] uses boosting to choose kernels. In the Gaussian process framework, Bayesian methods can be used to choose kernels [WB98, WR95]. If auxiliary data is known the EM algorithm can be used for learning the kernel [TAA03]. Finally, non stationary kernels were explored within the maximum entropy discrimination framework [LJN06].

These previous techniques focus on learning kernels which correspond to implicit mappings of the input features. Our technique learns explicit mappings and transformations. This allows prior knowledge to be exploited in the functional form of these mappings. Kernel based SVMs are often hard to interpret because they lead to complex decision boundaries which are only linear in the implicitly defined space. However, by working with explicit mappings, models with an intuitive meaning can be learned. The learned transformations can be visualized to lend insight into the problem, and the hyperplane weights indicate the importance of transformed variables. We next develop the mixture of transformations framework more formally and propose algorithms to efficiently find solutions.

Chapter 2

Large Margin Mixtures of Transformations

This chapter will formally present the large margin mixture of transformation learning problem and some simple algorithms for solving it. For an unknown distribution $P(\vec{x}, y)$ over inputs $\vec{x} \in \Re^D$ and labels $y \in \{-1, 1\}$, assume that there is an unknown transformation $\Phi(\vec{x})$, and an unknown hyperplane parameterized by $\vec{w}$ and b such that predicting with $f(\vec{x}) = \text{sign}(\vec{w}^T \Phi(\vec{x}) + b)$ yields a low expected test error $R = \int \frac{1}{2}|y - f(x)|dP(\vec{x}, y)$. The transformation, $\Phi(\vec{x}) = \sum_{j=1}^{J} m_j \phi_j(\vec{x})$, is defined by a set of basis transformations $\{\phi_1(\vec{x}), \ldots, \phi_J(\vec{x})\}$ with associated mixing weights $\{m_1, \ldots, m_J : m_j \in [0, 1], \sum_j m_j \leq 1\}$. Any linear combination of the transformations would give a new valid transformation, however the mixing weights need to be constrained so that the margin does not become infinite. The positivity constraint is used explicitly when learning monotonic transformations to impose monotonicity, however it is not needed in general to learn transformations. Maximum margin transformation learning addresses the task of recovering $\Phi(\vec{x}), \vec{w}, b$ from an input set of base transformations and a labeled training set $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ which is sampled i.i.d. from $P(\vec{x}, y)$. The method accomplishes this by learning a maximum margin hyperplane and the unknown transform $\Phi(\vec{x})$ simultaneously which leads to the following augmented SVM-like optimization:

$$\begin{aligned} \min_{\vec{w},b,\vec{\xi},\vec{m}} \quad & \|\vec{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i\left(\left\langle \vec{w}, \sum_{j=1}^{J} m_j\phi_j(\vec{x}_i)\right\rangle + b\right) \geq 1-\xi_i \,\forall i \\ & \xi_i \geq 0, m_j \geq 0, \sum_j m_j \leq 1 \,\forall i,j \end{aligned} \tag{2.1}$$

where $\vec{\xi}$ are the standard SVM slack variables and $\vec{w}$, b are the maximum margin solution for the training set that has been transformed via $\Phi(\vec{x})$ with learned $\vec{m}$, the vector of mixing weights.

This problem is nonconvex due to the quadratic term involving $\vec{w}$ and $\vec{m}$ in the classification constraints, and although it is difficult to find a globally optimal solution, the structure of the problem suggests a simple method for finding a locally optimal solution. The optimization can be divided into two convex subproblems that can be easily solved within a simple greedy coordinate descent algorithm. This algorithm iteratively solves for $\vec{w}, b$ and $\vec{\xi}$ with $\vec{\Phi}$ fixed by using an SVM solver to optimize:

$$\begin{aligned} \min_{\vec{w},b,\vec{\xi}} \quad & \|\vec{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i\left(\langle \vec{w}, \Phi(\vec{x}_i)\rangle + b\right) \geq 1-\xi_i,\ \xi_i \geq 0 \,\forall i. \end{aligned} \tag{2.2}$$

It then finds $\vec{m}$ and $\vec{\xi}$ with $\vec{w}$ and b fixed via a linear program:

$$\begin{aligned} \min_{\vec{m},\vec{\xi}} \quad & \sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i\left(\left\langle \vec{w}, \sum_{j=1}^{J} m_j\phi_j(\vec{x}_i)\right\rangle + b\right) \geq 1-\xi_i \,\forall i \\ & \xi_i \geq 0, m_j \geq 0, \sum_j m_j \leq 1 \,\forall i,j. \end{aligned} \tag{2.3}$$

Note that $\vec{\xi}$ is optimized in both subproblems as it is part of the hinge loss and not part of the model. This EM like algorithm quickly finds a locally optimal solution. Pseudo code for this optimization is listed as Algorithm 1.

Although the proposed greedy algorithm finds locally optimal solutions, the linear program has the unattractive characteristic of only minimizing the hinge loss while leaving

Algorithm 1 Greedy Large Margin Mixture of Transformations Learning

while not converged **do**

Solve C (or ν) SVM for $\vec{w}, b, \vec{\xi}$ (and ρ) with $\Phi(x)$ locked

Solve LP for $\Phi(x) = \sum_i m_i \phi_i(x), \vec{\xi}$ (and ρ) with $\vec{w}, b$ locked

end while

the margin unchanged. One way to address this is to use the ν-SVM algorithm [SSWB00] augmented for transformation learning:

$$\begin{aligned} \min_{\vec{w},b,\vec{\xi},\rho,\vec{m}} \quad & \|\vec{w}\|_2^2 - \nu\rho + \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i \left(\left\langle \vec{w}, \sum_{j=1}^{J} m_j \phi_j(\vec{x}_i) \right\rangle + b \right) \geq \rho - \xi_i \, \forall i \\ & \xi_i \geq 0, \rho \geq 0, m_j \geq 0, \sum_j m_j \leq 1 \, \forall i, j \end{aligned} \tag{2.4}$$

where ν takes the role of C in trading off between margin and accuracy. The previously proposed C-SVM formulation has a margin of $1/\|\vec{w}\|$ and the ν-SVM has a margin of $2\rho/\|\vec{w}\|$. The two formulations yield equivalent solutions when $\Phi(x)$ is fixed, $\rho > 0$ and C is set to $1/\rho$ a priori. The advantage of the ν-SVM formulation for transformation learning is that the linear program for learning the transformation weights maximizes the margin while minimizing the hinge loss thus (hopefully) finding better local solutions. The quadratic SVM used in the greedy iterative algorithm for this formulation with $\Phi(x)$ fixed is:

$$\begin{aligned} \min_{\vec{w},b,\vec{\xi},\rho} \quad & \|\vec{w}\|_2^2 - \nu\rho + \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i \left(\langle \vec{w}, \Phi(\vec{x}_i) \rangle + b \right) \geq \rho - \xi_i, \xi_i \geq 0, \rho \geq 0 \, \forall i. \end{aligned} \tag{2.5}$$

The linear program with $\vec{w}$ and b fixed is:

$$\begin{aligned} \min_{\vec{m},\vec{\xi},\rho} \quad & \sum_{i=1}^{N} \xi_i - \nu\rho \\ \text{subject to} \quad & y_i \left(\left\langle \vec{w}, \sum_{j=1}^{J} m_j \phi_j(\vec{x}_i) \right\rangle + b \right) \geq \rho - \xi_i \, \forall i \\ & \xi_i \geq 0, \rho \geq 0, m_j \geq 0, \sum_j m_j \leq 1 \, \forall i, j \end{aligned} \tag{2.6}$$

These two convex optimizations can again be solved iteratively as Algorithm 1 to find a locally optimal solution. This method now has the attractive property that both independent optimizations trade off between hinge loss and margin.

The models learned by these algorithms often predict well, but the methods get stuck in local minima. One way to mitigate this issue is to try multiple initializations. In practice, mixing weights $\vec{m}$, are initialized uniformly and alternative initializations do not yield much if any improvement. Due to these issues with local minima, it would be preferable to find an algorithm that can find global solutions, even if they are only approximate.

Chapter 3

Large Margin Matrix Mixtures of Transformations

3.1 Convex Relaxations

This chapter will look at learning large margin matrix mixtures of transformations. The matrix mixture will be derived as a convex relaxation of the previous transformation learning formulation. Although the locally optimal solutions found by the previous algorithms work well in practice, a method for finding global solutions would be preferred. When dealing with nonconvex quadratic terms in optimization problems, a typical approach is to relax these variables. This type of relaxation is well studied; it was originally proposed in [Sho87] and is the first in a sequence of convex relaxations introduced in [Las00]. It has been used to successfully approximate combinatorial problems such as Maximum Cut in [GW95] and to solve various machine learning problems such as sparse SVMs [CVL07] and transduction [BC04].

Convex relaxations for nonconvex quadratic problems can be derived in a number of different ways yielding the same relaxation. The simplest way to derive a relaxation is the use the Lagrangian relaxation or to derive the so-called semidefinite relaxation. These will yield the same relaxation and are in fact dual optimization problems. As a precursor to deriving a relaxation for the mixture of transformations, we will derive the semidefinite

relaxation for a general nonconvex quadratic optimization problem:

$$\begin{aligned} \min_{\vec{x}} \quad & \vec{x}^T A_0 \vec{x} + b_0^T \vec{x} \\ \text{subject to} \quad & \vec{x}^T A_i \vec{x} + b_i^T \vec{x} + c \leq 0, \ \forall i. \end{aligned} \tag{3.1}$$

First, the trace operator is introduced and $\vec{x}$ is grouped in the quadratic terms to form outer products, $\vec{x}\vec{x}^T$:

$$\begin{aligned} \min_{\vec{x}} \quad & \text{trace}(A_0 \vec{x}\vec{x}^T) + b_0^T \vec{x} \\ \text{subject to} \quad & \text{trace}(A_i \vec{x}\vec{x}^T) + b_i^T \vec{x} + c \leq 0, \ \forall i. \end{aligned} \tag{3.2}$$

The outer products $\vec{x}\vec{x}^T$ are then replaced with a matrix variable X and a nonconvex equality constraint, $X - \vec{x}\vec{x}^T = 0$, is introduced yielding an equivalent optimization:

$$\begin{aligned} \min_{\vec{x}} \quad & \text{trace}(A_0 X) + b_0^T \vec{x} \\ \text{subject to} \quad & \text{trace}(A_i X) + b_i^T \vec{x} + c \leq 0, \ \forall i \\ & X - \vec{x}\vec{x}^T = 0. \end{aligned} \tag{3.3}$$

The equality constraint, $X - \vec{x}\vec{x}^T = 0$, is replaced with a semidefinite constraint $X - \vec{x}\vec{x}^T \succeq 0$ yielding a convex relaxation:

$$\begin{aligned} \min_{\vec{x}} \quad & \text{trace}(A_0 X) + b_0^T \vec{x} \\ \text{subject to} \quad & \text{trace}(A_i X) + b_i^T \vec{x} + c \leq 0, \ \forall i \\ & X - \vec{x}\vec{x}^T \succeq 0. \end{aligned} \tag{3.4}$$

In order to convert this into a semidefinite program, the Schur complement lemma must be invoked:

$$X - \vec{x}\vec{x}^T \succeq 0 \iff \begin{pmatrix} 1 & \vec{x}^T \\ \vec{x} & X \end{pmatrix} \succeq 0. \tag{3.5}$$

Substituting this result producers the following SDP:

$$\begin{aligned} \min_{\vec{x}} \quad & \text{trace}(A_0 X) + b_0^T \vec{x} \\ \text{subject to} \quad & \text{trace}(A_i X) + b_i^T \vec{x} + c \leq 0, \ \forall i \\ & \begin{pmatrix} 1 & \vec{x}^T \\ \vec{x} & X \end{pmatrix} \succeq 0. \end{aligned} \tag{3.6}$$

This derivation yields a general recipe for deriving convex relaxations for nonconvex quadratic optimizations which can be solved with standard SDP interior points algorithms.

3.2 A Convex Algorithm for Learning Matrix Mixture of Transformations

We now show how to apply convex semidefinite relaxations to the large margin mixture of transformations formulation in an efficient manner. The original nonconvex optimization problem introduced in Chapter 2,

$$\begin{aligned} \min_{\vec{w},\vec{\xi},b,\vec{m}} \quad & \|\vec{w}\|_2^2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i \left(\left\langle \vec{w}, \sum_{j=1}^{J} m_j \phi_j(\vec{x}_i) \right\rangle + b \right) \geq 1 - \xi_i \, \forall i \\ & \xi_i \geq 0, m_j \geq 0, \sum_j m_j \leq 1 \, \forall i, j, \end{aligned} \tag{3.7}$$

is quadratic in both $\vec{w}$ and $\vec{m}$. If these terms are relaxed using the recipe from the previous section, it would result in a $(D + J + 1) \times (D + J + 1)$ semidefinite matrix variable where D is the dimension of the mapped data and J is the number of transformations. This is problematic because the relaxed matrix grows with both the number of transformations and the dimension of the data. Many of the learning tasks that are addressed later in this dissertation involve high dimensional data.

With the proper reformulation, a relaxation can be derived so that only quadratic terms involving $\vec{m}$ need to be replaced with entries in a semidefinite matrix. This yields a tighter relaxation and avoids excessive computational costs involved with optimizing a matrix that scales with the dimension of the data. In order to derive this improved relaxation, the

problem must first be transformed into the dual with respect to $\vec{w}$, $\vec{\xi}$, and b similar to the dual SVM. First, for a fixed $\vec{m}$, the Lagrangian function for the optimization in (3.7) is introduced:

$$\mathcal{L}(\vec{w},\vec{\xi},b,\vec{\alpha},\vec{\beta}) = \|\vec{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i - \sum_i \alpha_i \left(y_i \left(\left\langle \vec{w}, \sum_{j=1}^{M} m_j \phi_j(\vec{x}_i) \right\rangle + b \right) - 1 + \xi_i \right) - \sum_i \beta_i \xi_i. \tag{3.8}$$

The Lagrange dual is then defined by minimizing the Lagrange function over the primal variables, $\min_{\vec{w},\vec{\xi},b} \mathcal{L}(\vec{w},\vec{\xi},b,\vec{\alpha},\vec{\beta})$. Because the initial problem (with $\vec{m}$ fixed) is convex, maximizing the Lagrange dual function over the dual variables, $\vec{\alpha}$ and $\vec{\beta}$ yields the same solution as the original problem. To eliminate the primal variables, $\vec{w}$, $\vec{\xi}$, b, from this dual formulation, the corresponding partial derivatives of $\mathcal{L}$ are computed and set to 0. These optimality conditions are used to derive the dual optimization:

$$\begin{aligned} \min_{\vec{m}} \max_{\vec{\alpha}} \quad & 2\vec{\alpha}^T\vec{1} - \vec{\alpha}^T \left(Y \left(\sum_{i,j} m_i m_j K_{i,j} \right) Y \right) \vec{\alpha} \\ \text{subject to} \quad & \vec{0} \le \vec{\alpha} \le C\vec{1}, \vec{\alpha}^T\vec{y} = 0, \vec{m} \ge \vec{0}, \vec{m}^T\vec{1} \le 1 \end{aligned} \tag{3.9}$$

where $K_{i,j} = \phi_i(X)^T\phi_j(X)$ is the inner product matrix of transformed examples. The mixing weights $\vec{m}$ are then relaxed by substituting the matrix variable M by using the recipe from the previous section. An additional (seemingly) redundant constraint $\sum_j m_j^2 \le 1$ can be added to ensure a tighter relaxation:

$$\begin{aligned} \min_{M} \max_{\vec{\alpha}} \quad & S(M,\alpha) = 2\vec{\alpha}^T\vec{1} - \vec{\alpha}^T \left(Y \left(\sum_{i,j} M_{i,j} K_{i,j} \right) Y \right) \vec{\alpha} \\ \text{subject to} \quad & M \succeq 0, \sum_{j=1}^{J} M_{j,j} \le 1, M_{0,i} \ge 0, \sum_{j=1}^{J} M_{0,j} \le 1, M_{0,0} = 1 \; \forall i \\ & \vec{0} \le \vec{\alpha} \le C\vec{1}, \vec{\alpha}^T\vec{y} = 0. \end{aligned} \tag{3.10}$$

Equation 3.10 is a saddle point problem whose solution, which is equivalent with the min and max reversed: $\min_M \max_{\vec{\alpha}} S(M,\alpha) = \max_{\vec{\alpha}} \min_M S(M,\alpha)$, is known as a saddle value.

The equality when interchanging the minimization and maximization and the existence of the saddle value are due to Sion's minimax theorem [Sio58] which is a generalization of the Von Neumann minimax theorem [Neu28]. Sion's theorem can be stated as in [Kom88]:

Theorem 3 *Let X be a compact convex subset of a linear topological space and Y a convex subset of a linear topological space. Let f be a real-valued function on $X \times Y$ such that:*

(i) *$f(x, \cdot)$ is upper semicontinuous and quasi-concave on Y for each $x \in X$*

(ii) *$f(\cdot, y)$ is lower semicontinuous and quasi-convex on X for each $y \in Y$*

Then $\min_{x \in X} \sup_{y \in Y} f(x, y) = \sup_{y \in Y} \min_{x \in X} f(x, y)$.

$S(M, \alpha)$ fits these criteria because it is linear in M and quadratic in α. Additionally, M is constrained to be in a convex set and α is constrained to be in a compact convex set.

Sometimes in kernel learning, it is possible to exploit additional structure in the problem by rearranging the min and max to yield a simpler optimization problem. Learning a convex combination of kernels [LCB+04] and learning sequence kernels [CMR08] are examples where a either a QCQP or QP (respectively) can be solved instead of the more general SDP formulation through this rearrangement. Unfortunately in the large margin transformation learning problem, such a simple modification does not yield additional structure that can be exploited for a simpler algorithm.

Working with Equation 3.10, the next step will be to transform this constrained saddle point problem into a semidefinite program that can be solved with standard solvers. The dual formulation with respect to $\vec{\alpha}$ can be found by first defining a new Lagrangian with M held fixed:

$$\mathcal{L}(\vec{\alpha}, \lambda, \vec{\nu}, \vec{\delta}) = 2\vec{\alpha}^T\vec{1} - \vec{\alpha}^T \left(Y \left(\sum_{i,j} M_{i,j} K_{i,j} \right) Y \right) \vec{\alpha} + 2\vec{\nu}^T\vec{\alpha} + \lambda\vec{y}^T\vec{\alpha} + 2\vec{\delta}^T(C\vec{1} - \vec{\alpha}) \quad (3.11)$$

where λ, $\vec{\nu}$, $\vec{\delta}$ are dual variables. The Lagrangian dual function $\max_\alpha \mathcal{L}(\vec{\alpha}, \lambda, \vec{\nu}, \vec{\delta})$ can be minimized over the dual variables to yield an alternative equivalent optimization. Again, in order to eliminate $\vec{\alpha}$, the gradient of $\mathcal{L}(\vec{\alpha}, \lambda, \vec{\nu}, \vec{\delta})$ with respect to α is computed and set to 0. Solving this optimality condition for $\vec{\alpha}$ yields:

$$\vec{\alpha} = \left(Y \left(\sum_{i,j} M_{i,j} K_{i,j} \right) Y \right)^{-1} \left(\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \right). \tag{3.12}$$

Substituting for $\vec{\alpha}$ yields a minimization problem over the dual variables and M:

$$\begin{aligned} \min_{M,\lambda,\vec{\nu},\vec{\delta}} \quad & \left(\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \right)^T \left(Y \left(\sum_{i,j} M_{i,j} K_{i,j} \right) Y \right)^{-1} \left(\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \right) + 2C \vec{\delta}^t \vec{1} \\ \text{s.t.} \quad & M \succeq 0, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, \sum_{j=1}^{J} M_{0,j} \leq 1, M_{0,0} = 1, \vec{\nu} \geq \vec{0}, \vec{\delta} \geq \vec{0}, \forall i. \end{aligned}$$

The cost function can be transformed into a constraint yielding a linear cost:

$$\begin{aligned} \min_{M,\lambda,\vec{\nu},\vec{\delta},t} \quad & t \\ \text{s.t.} \quad & \left(\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \right)^T \left(Y \left(\sum_{i,j} M_{i,j} K_{i,j} \right) Y \right)^{-1} \left(\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \right) + 2C \vec{\delta}^t \vec{1} \leq t \\ & M \succeq 0, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, \sum_{j=1}^{J} M_{0,j} \leq 1, M_{0,0} = 1, \vec{\nu} \geq \vec{0}, \vec{\delta} \geq \vec{0}, \forall i. \end{aligned}$$

In order to convert this optimization problem into a semidefinite program, the Schur complement lemma can be used to transform the quadratic inequality constraint into a semidefinite constraint. For $Y \sum_{i,j} M_{i,j} K_{i,j} Y \succ 0$,

$$\begin{aligned} & t - 2C \vec{\delta}^t \vec{1} - \left(\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \right)^T \left(Y \left(\sum_{i,j} M_{i,j} K_{i,j} \right) Y \right)^{-1} \left(\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \right) \geq 0 \\ \iff & \begin{pmatrix} Y \sum_{i,j} M_{i,j} K_{i,j} Y & \vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \\ (\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y})^T & t - 2C \vec{\delta}^T \vec{1} \end{pmatrix} \succeq 0. \end{aligned}$$

If $Y \sum_{i,j} M_{i,j} K_{i,j} Y$ is not positive definite, a small value can be added to the diagonal to increase any eigenvalues that are zero. Substituting this back into the optimization results in a positive semidefinite program similar to the multiple kernel learning framework [LCB+04].

$$\begin{aligned} \min_{M,t,\lambda,\vec{\nu},\vec{\delta}} \quad & t \\ \text{subject to} \quad & \begin{pmatrix} Y \sum_{i,j} M_{i,j} K_{i,j} Y & \vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y} \\ (\vec{1} + \vec{\nu} - \vec{\delta} + \lambda \vec{y})^T & t - 2C\vec{\delta}^T \vec{1} \end{pmatrix} \succeq 0 \\ & M \succeq 0, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, \sum_{j=1}^{J} M_{0,j} \leq 1, M_{0,0} = 1, \vec{\nu} \geq \vec{0}, \vec{\delta} \geq \vec{0}, \forall i. \end{aligned}$$

This SDP defines the matrix mixture of transformations problem. If $\text{rank}(M) = 1$ then the relaxation is exact and reduces to the standard mixture of transformations. Additionally, if M is a diagonal matrix it reduces to a mixture of kernels [LCB+04].

Given that this is a relaxation and that there are no explicit constraints enforcing the positive semidefiniteness of the kernel, it needs to be shown that kernels learned in this manner are indeed positive semidefinite. To this end, we prove the following theorem.

Theorem 4 *Let M be a positive semidefinite matrix and $\{\phi_1(x), ..., \phi_N(x)\}$ be a set of mappings from $x \in R^n$ to a common space Γ. A combination $k(x, x') = \sum_{i,j} M_{i,j} \phi_i(x)^T \phi_j(x')$ yields a Mercer kernel.*

Proof 5 *$k(x, x')$ is a Mercer kernel if and only if $\int k(x, x') g(x) g(x') dx dx' \geq 0$. By definition we have that:*

$$S^k = \int \sum_{i,j} M_{i,j} \phi_i(x)^T \phi_j(x') g(x) g(x') dx dx' \tag{3.13}$$

Rewrite the positive semidefinite matrix as $M = VDV^T$ in terms of an eigenvector matrix V and a diagonal matrix of eigenvalues $D = diag([d_1, \ldots, d_n])$.

$$S^k = \int \sum_{i,j} \sum_{k} d_k V_{i,k} V_{j,k} \phi_i(x)^T \phi_j(x') g(x) g(x') dx dx' \tag{3.14}$$

which factorizes as:

$$S^k = \sum_k d_k \left(\int \sum_i V_{i,k} g(x) \phi_i(x)^T dx \right) \left(\int \sum_j V_{j,k} g(x) \phi_j(x') dx' \right). \tag{3.15}$$

The two factors are actually the same and can be expressed as one quadratic term:

$$\sum_k d_k \left\| \int \sum_i V_{i,k} g(x) \phi_i(x) dx \right\|^2 \geq 0. \tag{3.16}$$

This conic combination is always positive due to the positivity of both the eigenvalues d_k and the norm.

Learning matrix mixtures of transformations can be analyzed using the generalization bounds derived for learning mixtures of kernels [].

Theorem 6 *Given a finite set of J mappings $\{\phi_1(x), \phi_2(x), ..., \phi_J(x)\}$ and a learned kernel defined by $k(x, x') = \sum_{i,j} M_{i,j} k_{i,j}(x, x')$ where M is the learned mixing matrix and $k_{i,j}(x, x') = \phi_i(x)^T \phi(x')$ the estimation error is bounded by $\sqrt{\tilde{O}(\frac{J^2}{2} + J + \frac{1}{\gamma^2})/N}$ where N is the sample size and $\tilde{O}()$ notation hides logarithmic factors as well as some terms involving the sample size and the failure probability.*

Proof 7 *$\sqrt{\tilde{O}(d_p + \frac{1}{\gamma^2})/N}$ is the bound on the estimation error for a learned kernel with pseudodimension d_p []. What remains to prove is that a learned kernel function of the form $k(x, x') = \sum_{i,j} M_{i,j} k_{i,j}(x, x')$ has $d_p = \frac{J^2}{2} + J$. $k(x, x')$ can be seen to be an element of a vector space defined by the $\frac{J^2}{2} + J$ basis functions $\{k_{i,j}(x, x') + k_{j,i}(x, x') | 1 \leq i \leq J, 1 \leq j < i\}$. The matrix M takes linear combinations of these basis functions. The pseudodimension of $k(x, x')$ is then bounded by $\frac{J^2}{2} + J$ because the pseudodimension of a space of real valued functions is bounded by its dimension [].*

This proof holds for any matrix M so it is valid for both the mixture of transformations whose mixing matrix is $M = \vec{m}\vec{m}^T$ and for semidefinite matrices learned in the matrix mixture of transformation framework. This bound does not take into account that M is positive definite or that most learned matrices are low rank in the matrix mixture of transformations framework and rank one for mixtures of transformations. Low rank matrices yield simpler models but it is not clear how to relate the rank of mixture matrices to the pseudodimension. This would be an interesting topic for future exploration.

Chapter 4

Extragradient Based Transformation Learning

4.1 The Extragradient Algorithm

It is well known that semidefinite programs scale poorly with interior point solvers scaling as $O\left(q^2(\sum_i p_i^2)\sqrt{p}\right)$ [VB96] where q is the number of variables, p_i is the size of the ith semidefinite block and $p = \sum_i p_i$. Due to this poor scaling, an alternative algorithm is necessary to work with larger data sets and to extend the model efficiently to more complicated scenarios. The semidefinite program for finding large margin matrix mixture of transformations,

$$\begin{aligned} \min_{M,t,\lambda,\vec{\nu},\vec{\delta}} \quad & t \\ \text{subject to} \quad & \begin{pmatrix} Y\sum_{i,j} M_{i,j}\phi_i(X)^T\phi_j(X)Y & \vec{1}+\vec{\nu}-\vec{\delta}+\lambda\vec{y} \\ (\vec{1}+\vec{\nu}-\vec{\delta}+\lambda\vec{y})^T & t-2C\vec{\delta}^T\vec{1} \end{pmatrix} \succeq 0 \\ & M \succeq 0, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, \sum_{j=1}^{J} M_{0,j} \leq 1, M_{0,0} = 1, \vec{\nu} \geq \vec{0}, \vec{\delta} \geq \vec{0}, \forall i, \end{aligned}$$

has two linear matrix inequalities which are the main computational bottleneck. The first constraint,

$$\begin{pmatrix} Y\sum_{i,j} M_{i,j}\phi_i(X)^T\phi_j(X)Y & \vec{1}+\vec{\nu}-\vec{\delta}+\lambda\vec{y} \\ (\vec{1}+\vec{\nu}-\vec{\delta}+\lambda\vec{y})^T & t-2C\vec{\delta}^T\vec{1} \end{pmatrix} \succeq 0, \tag{4.1}$$

scales with the number of data points and is problematic in a data rich machine learning setting. The second semidefinite constraint on the mixing matrix, $M \succeq 0$, scales with the number of transformations which is typically a small number and therefore is not as problematic. The scaling for large margin matrix mixture of transformations is $O\left(q^2(\sum_i p_i^2)\sqrt{p}\right)$ where $q = 2N + \frac{(J+1)^2}{2} + 2$, $p_1 = N+1$, $p_2 = J+1$ and $p = N+J+2$ where N is the number of data points and J is the number of transformations. The constraint associated with p_1, which depends on the number of data points, arises when the saddle point problem,

$$\begin{aligned} \min_{M} \max_{\vec{\alpha}} \quad & S(M,\alpha) = 2\vec{\alpha}^T\vec{1} - \vec{\alpha}^T \left(Y \left(\sum_{i,j} M_{i,j}\phi_i(X)^T\phi_j(X) \right) Y \right) \vec{\alpha} \\ \text{subject to} \quad & M \succeq 0, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, \sum_{j=1}^{J} M_{0,j} \leq 1, M_{0,0} = 1 \; \forall i \\ & \vec{0} \leq \vec{\alpha} \leq C\vec{1}, \vec{\alpha}^T\vec{y} = 0. \end{aligned} \tag{4.2}$$

is transformed into an the SDP via the Schur complement. Instead the saddle point problem, $S(M,\alpha)$, can be optimized directly thereby eliminating the large semidefinite constraint.

The extragradient algorithm [Kor76, Kho89] is a first order method for solving saddle point problems (and a larger class of problems called variational inequalities). It has previously been used in machine learning to efficiently solve structured learning problems [TLJJ05]. The algorithm uses only gradient information and consists of a prediction and correction step. The step size of each is tuned automatically. After each step, the new solution is projected back to the convex constraint set. If there are efficient algorithms for computing these projections, each step can be computed quickly which is imperative for good performance. In Section 4.2, we show how to compute these projections efficiently.

The algorithm proceeds as follows and iterates until a stopping criterion is reached:

$$\text{(Prediction)} \quad \begin{cases} \bar{M}^{k+1} = P_M\left(M^k - \beta_k \nabla_M S(M^k, \alpha^k)\right) \\ \bar{\alpha}^{k+1} = P_\alpha\left(\alpha^k + \beta_k \nabla_\alpha S(M^k, \alpha^k)\right) \end{cases} \tag{4.3}$$

$$\text{(Correction)} \quad \begin{cases} M^{k+1} = P_M\left(M^k - \beta_k \nabla_M S(\bar{M}^{k+1}, \bar{\alpha}^{k+1})\right) \\ \alpha^{k+1} = P_\alpha\left(\alpha^k + \beta_k \nabla_\alpha S(\bar{M}^{k+1}, \bar{\alpha}^{k+1})\right) \end{cases} \tag{4.4}$$

where P_M and P_α are projection operators, ∇_M and ∇_α are gradient operators and β_k is the step size at iteration k. The required gradients for this problem are:

$$\nabla_{M_{i,j}} S(M,\alpha) = \vec{\alpha}^T \left(Y \left(\phi_i(X)^T \phi_j(X) \right) Y \right) \vec{\alpha}, \tag{4.5}$$

$$\nabla_{\alpha} S(M,\alpha) = 2\vec{1} + 2 \left(Y \left(\sum_{i,j} M_{i,j} \phi_i(X)^T \phi_j(X) \right) Y \right) \vec{\alpha}. \tag{4.6}$$

The projections that must be efficiently solved are:

$$\begin{aligned} P_M(M) = \min_{\hat{M}} \quad & \|\hat{M} - M\|_F^2 \\ \text{subject to:} \quad & \hat{M} \succeq 0, \sum_{j=1}^{J} \hat{M}_{j,j} \leq 1, \hat{M}_{0,i} \geq 0, \sum_{j=1}^{J} \hat{M}_{0,j} \vec{1} \leq 1, \hat{M}_{0,0} = 1, \ \forall i \end{aligned} \tag{4.7}$$

$$\begin{aligned} P_\alpha(\vec{\alpha}) = \min_{\hat{\alpha}} \quad & \|\hat{\alpha} - \vec{\alpha}\|_2^2 \\ \text{subject to:} \quad & 0 \leq \hat{\alpha}_i \leq C. \end{aligned} \tag{4.8}$$

In the extragradient algorithm, the step size is automatically tuned by first computing

$$r_k = \beta_k \frac{\nabla S(M^k, \alpha^k) - \nabla S(\bar{M}^{k+1}, \bar{\alpha}^{k+1})\|}{(\|M^k - \bar{M}^{k+1}\| + \|\alpha^k - \bar{\alpha}^{k+1}\|)}. \tag{4.9}$$

Then, if $r_k > \nu$, where $\nu \in (0,1)$ is a parameter of the algorithm, β_k is decreased via an Armijo type rule: $\beta_k = (2/3)\beta_k \min(1, 1/r_k)$ and a new prediction step is computed. Pseudocode for the extragradient algorithm applied to large margin matrix mixture transformation learning can be found as Algorithm 2. Further speedups to the basic extragradient algorithm can be found in [HL02].

There are a number of advantages to using the extragradient algorithm. By working with the saddle point problem (4.2) directly, the semidefinite constraint (4.1), which grows with the number of data points, is avoided. Another way to improve training time with the extragradient algorithm is to initialize it with previous solutions that are often available when cross validating over parameters in a machine learning setting. Semidefinite programs cannot make use of these warm start initializations. Additionally, the extragradient algorithm also permits early stopping which can also improve training time. Machine learning algorithms do not need to learn numerically exact models, only predictively accurate ones. So early stopping is often an appropriate strategy. In practice, the algorithm is stopped when the cost function changes by less than ϵ.

Algorithm 2 Extragradient Algorithm

$k = 0$

while not converged **do**

repeat

Prediction:

$\bar{M}^{k+1} = P_M\left(M^k - \beta \nabla_M S(M^k, \alpha^k)\right)$

$\bar{\alpha}^{k+1} = P_\alpha\left(\alpha^k + \beta \nabla_\alpha S(M^k, \alpha^k)\right)$

$r = \beta \frac{\|\nabla S(M^k, \alpha^k) - \nabla S(\bar{M}^{k+1}, \bar{\alpha}^{k+1})\|}{(\|M^k - \bar{M}^{k+1}\| + \|\alpha^k - \bar{\alpha}^{k+1}\|)}$

if $r \geq \nu$ **then**

$\beta = (2/3)\beta \min(1, 1/r)$

end if

until $r < \nu$

Correction:

$M^{k+1} = P_M\left(M^k - \beta \nabla_M S(\bar{M}^{k+1}, \bar{\alpha}^{k+1})\right)$

$\alpha^{k+1} = P_\alpha\left(\alpha^k + \beta \nabla_\alpha S(\bar{M}^{k+1}, \bar{\alpha}^{k+1})\right)$

$k = k + 1$

end while

One additionally technicality involving the bias term, b, needs to be addressed. When solving the SDP, the bias term is recovered from the KKT conditions. However, the extragradient algorithm is typically stopped before exact convergence, so computing the bias term from the KKT conditions becomes unstable. Therefore, the bias term is dropped from the model and if a bias term is desired, an additional dimension of all ones can be added to the training data. Empirically adding an additional dimension to represent the bias affects the accuracy very little. A beneficial side effect from dropping the bias term is that the constraint $\vec{\alpha}^T\vec{y} = 0$ is eliminated which allows the projection for $\vec{\alpha}$ to be computed in closed form.

4.2 Dykstra's Iterative Projection Algorithm

The key to a fast implementation of the extragradient algorithm is to compute the projection steps with efficient algorithms. While the projection for $\vec{\alpha}$ (4.8) can be simply computed by clamping values between 0 and C, the projections for M (4.7) cannot be solved in closed form or with simple algorithms. A naive implementation would be to use a general purpose SDP solver to compute the projection, but this proves to be too computationally burdensome. However, if a convex set can be represented as an intersection of simpler convex sets, Dykstra's iterative projection algorithm [Dyk83] can be used to solve the full projection by iteratively projecting onto the simpler sets. This can be a very efficient algorithm if projections on the simple sets can be computed quickly. The convex set for M is amenable to such efficiently computable projections.

For a convex set $\mathcal{C} = \cap_{r=0}^{R-1} C_r$, that can be split into the intersection of a set of R simple convex sets $\{C_0, C_2, \ldots, C_{R-1}\}$, Dykstra's algorithm computes $P_{\mathcal{C}}(x)$, the projection of x onto $\mathcal{C}$ by iteratively computing $P_{\mathcal{C}_{[n]}}(x_{n-1} + e_{[n]})$, the projection onto $\mathcal{C}_{[n]}$ of the previous solution, x_{n-1}, plus a residual term $e_{[n]}$ where $[n]$ is defined as $n \bmod R$. If the convex sets are affine, the residual terms are not needed. Dykstra's algorithm is presented as Algorithm 3 and can be visualized in Figure 4.1.

Dykstra's method is the most popular algorithm for solving the best approximation problem in an intersection of convex sets. Other projection based methods for solving

Algorithm 3 Dykstra's Iterative Projection Algorithm

$x_0 = x, e_r = ... = e_1 = e_0 = 0$

$n = 0$

while not converged **do**

$n = n + 1$

$x_n = P_{\mathcal{C}_{[n]}}(x_{n-1} + e_{[n]})$

$e_{[n]} = x_{n-1} + e_{[n]} - x_n$

end while

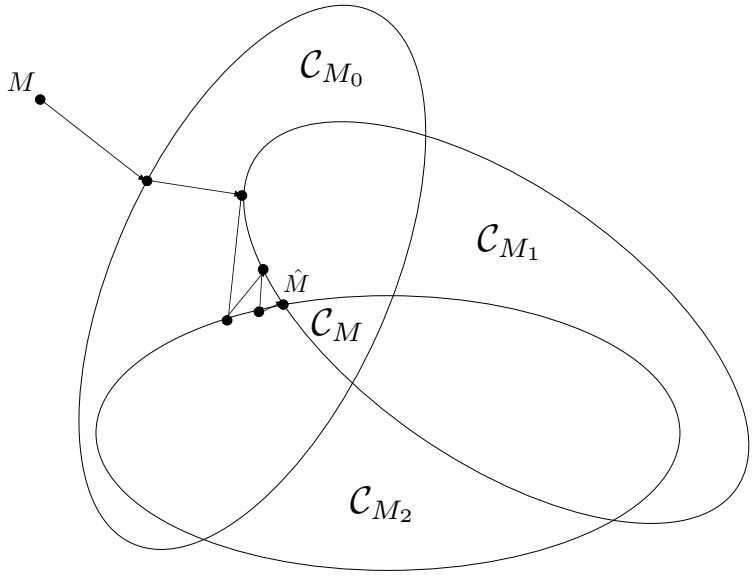


Figure 4.1: Dykstra's iterative projection algorithm used to solve the projection $P_M(M) = \min_{\hat{M} \in \mathcal{C}_M} \|\hat{M} - M\|_F^2$ where $\mathcal{C}_M$ is the intersection of three simple convex sets: $\mathcal{C}_M = \mathcal{C}_{M_0} \cap \mathcal{C}_{M_1} \cap \mathcal{C}_{M_2}$.

this problem include Haugazeau's method [BC01] and an algorithm by Bauschke [Bau96]. Dykstra's method was shown to converge linearly for the case of an intersection of half spaces [DH94, Shu00], namely that:

$$\|x_n - P_{\mathcal{C}}(x)\| \leq \rho c^n \tag{4.10}$$

where ρ and c are constants with $\rho > 0$ and $0 \leq c \leq 1$. Unfortunately this convergence rate cannot be applied in our setting because large margin transformation learning uses more complicated convex sets than simple half spaces.

For the large margin matrix mixture of transformations saddle point formulation, the convex sets for M are defined as:

$$\mathcal{C}_M \quad = \quad \mathcal{C}_{M_0} \cap \mathcal{C}_{M_1} \cap \mathcal{C}_{M_2},$$

the intersection of three simple convex sets:

$$\begin{aligned}
\mathcal{C}_{M_0} &= \{M : M \in \Re^{N+1 \times N+1}, M \succeq 0\} \\
\mathcal{C}_{M_1} &= \{M : M \in \Re^{N+1 \times N+1}, M = M^T, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, M_{0,0} = 1\} \forall i \\
\mathcal{C}_{M_2} &= \{M : M \in \Re^{N+1 \times N+1}, \sum_{j=1}^{J} M_{0,j} \leq 1\}.
\end{aligned}$$

The projection $P_{M_0}(M) = \min_{\hat{M}} \|\hat{M} - M\|_F^2$ s.t. $\hat{M} \succeq 0$ can be computed by finding the eigenvalue decomposition, zeroing out any negative eigenvalues and recomposing the matrix,

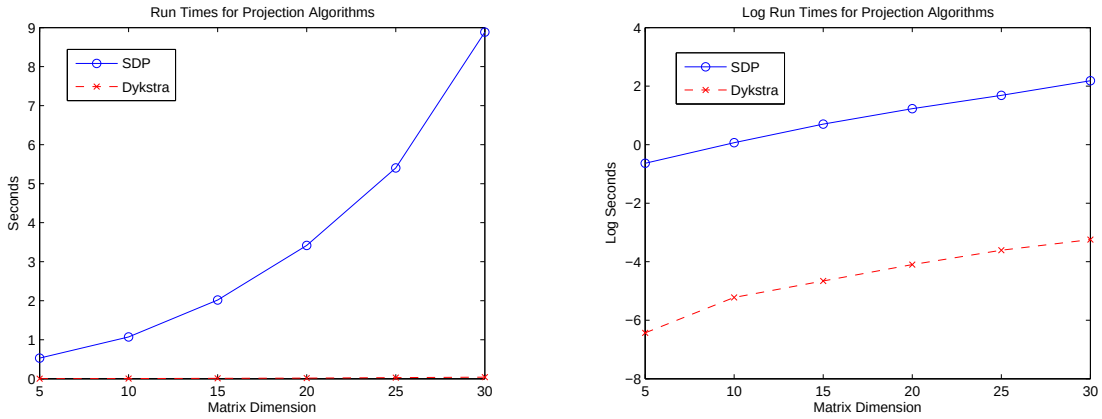
$$M = VDV^T, \ \hat{M} = V \max(D, 0)V^T. \tag{4.11}$$

The projection $P_{M_1}(M) = \min_{\hat{M}} \|\hat{M} - M\|_F^2, \ M = M^T, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, M_{0,0} = 1$ can be computed by averaging M and M^T and zeroing any negative entries

$$\hat{M} = \max((M + M^T)/2, 0), \tag{4.12}$$

then setting $\hat{M}_{0,0} = 1$ and updating entries in the diagonal if $\sum_{j=1}^{J} M_{j,j} > 1$ with a projection onto a halfspace:

$$\hat{M}_{i,i} = M_{i,i} + \frac{1 - \sum_{j=1}^{J} M_{j,j}}{J}. \tag{4.13}$$



(a) Running time (b) Log running time

Figure 4.2: Running time for projection algorithms. The running time for Dykstra's iterative projection algorithm (dashed red) is compared to the equivalent SDP (solid blue) for various matrix dimensions averaged over 10 random matrices. The left image shows that Dykstra's algorithm solves the projection two orders of magnitude faster. The log plot on the right shows that the two algorithms scale similarly.

The projection $P_{M_2} = \min_{\hat{M}} \|\hat{M} - M\|_F^2$ s.t. $\sum_{j=1}^{J} M_{0,j} \leq 1$ can be computed as a projection onto a halfplane if $\sum_{j=1}^{J} M_{0,j} > 1$:

$$\hat{M}_{0,i} = M_{0,i} + \frac{1 - \sum_{j=1}^{J} M_{0,j}}{J}. \tag{4.14}$$

Generic SDP solvers could be used in place of Dykstra's algorithm to solve these projections at the cost of more compute time. Figure 4.2(a) shows the run times for computing projections of a $D \times D$ matrix using a semidefinite program (blue) and Dykstra's algorithm (red). Dykstra's algorithm solves the projection two orders of magnitude faster than the SDP. Figure 4.2(b) shows the log of runtime demonstrating that the two algorithms scale similarly.

Not only does Dykstra's algorithm accelerate the saddle point optimization of Equation (4.2), fast projections allow the extragradient algorithm to be applied to a wider variety of applications. In the following section we make use of the extragradient algorithm and Dykstra's algorithm in a multitask setting. Additional extensions can be further explored with this efficient algorithm.

Chapter 5

Multiple Task Transformation Learning

A common theme in machine learning is to try to use all possible sources of information to learn the most accurate model possible. This information could be in the form of prior knowledge which can be encoded in the kernel or the function class being learned. It could be in the form of unlabeled data which yields a semi-supervised [CSZ06] or transductive algorithm [Vap98, Joa99]. Another approach is to leverage information from related tasks [AZ05, EP04, Jeb04]. This final setting is the so called multiple task (multitask) learning framework and is the subject of this chapter in the context of transformation learning.

We start by defining a formal definition of the multitask learning framework. Assume there are T related learning tasks defined on the same input domain each with labeled training sets $S_t = \{(\vec{x}_{1,t}, y_{1,t}), \ldots, (\vec{x}_{N_t,t}, y_{N_t,t})\}$. The goal in multitask learning is to learn a joint structure for all tasks that will allow a sharing of information based on task similarity. A joint optimization can be formulated where each task searches for a target function $f_l \in \mathcal{H}_{t,\theta}$ in its own task specific hypothesis space $\mathcal{H}_{t,\theta}$ which has a shared parameter, θ, common to all tasks:

$$\min_{\theta, f_t \in \mathcal{H}_{t,\theta}} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} L(f_t(x_{i,t}, y_{i,t})) \tag{5.1}$$

where L is an arbitrary loss function. This basic framework was proposed under the name

of structure learning [AZ05]. For the support vector machine, L is the hinge loss and a number of different shared structures have been explored. Perhaps the simplest common structure would be to the same C and kernel parameters for all of the tasks. Another way to tie tasks together is to estimate a mean function $f_0(x)$ and make predictions for task t with $f_0(x) + f_t(x)$ [EP04]. More complicated structures such as learning a common feature selection or a common kernel have also been explored [Jeb04].

We are interested in the case where the shared structure is a mixture of transformations $\Phi(x) = \sum_i m_i \phi_i(x)$ or a matrix mixture of transformations defining a kernel as $K(x, x') = \sum_{i,j} M_{i,j} \phi_i(x) \phi_j(x')$. This couples the problems and allows for the transfer of information between related tasks in the form of these joint transformations. The joint optimization can be phrased as the following nonconvex optimization:

$$\begin{aligned} \min_{\vec{w}_{1,..,T}, \vec{\xi}_{1,..,T}, b_{1,..,T}, \vec{m}} \quad & \sum_{t=1}^{T} \left(\|\vec{w}_t\|_2^2 + C \sum_{i=1}^{N_t} \xi_{i,t} \right) \\ \text{subject to} \quad & y_{i,t} \left(\left\langle \vec{w}_t, \sum_{j=1}^{M} m_j \phi_j(\vec{x}_{i,t}) \right\rangle + b_t \right) \geq 1 - \xi_{i,t} \\ & \xi_{i,t} \geq 0, m_j \geq 0, \sum_j m_j \leq 1. \end{aligned}$$

Similar to the single task case, a simple greedy algorithm can be derived to solve this initial nonconvex formulation. The task specific hyperplanes, parameterized with (w_t, b_t) can be optimized independently with an SVM solver when the transformation $\Phi(x) = \sum_i m_i \phi_i(x)$ held fixed. $\Phi(x)$ can be found with a linear program with the hyperplanes held fixed. This alternating procedure can be run to convergence to find a locally optimal solution and is described in Algorithm 3.

Following the derivations of the single task setting, a relaxation can be derived for the multitask mixture of transformations problem. This will again yield the matrix mixture of transformations with a single kernel for all tasks defined as $K(x, x') = \sum_{i,j} M_{i,j} \phi_i(x)^T \phi_j(x')$. Similar to Equation (3.10), the dual can be derived and $\vec{m}$ relaxed yielding another saddle point optimization:

Algorithm 4 Multitask Greedy Mixture of Transformation Learning

while not converged **do**

for $t = 1$ to T **do**

Solve SVM for $\vec{w}_t, b_t, \vec{\xi}_t$ with $\Phi(x)$ locked

end for

Solve LP for $\Phi(x) = \sum_i m_i \phi_i(x), \{\vec{\xi_1}, \ldots, \vec{\xi_T}\}$ with $\{(\vec{w}_1, b_1), \ldots, (\vec{w}_T, b_T)\}$ locked

end while

$$\begin{aligned} \min_{M} \max_{\vec{\alpha}_{1,..,T}} \quad & \sum_{t=1}^{T} \left(2\vec{\alpha_t}^T \vec{1} - \vec{\alpha_t}^T \left(Y_t \left(\sum_{i,j} M_{i,j} \phi_i(X_t)^T \phi_j(X_t) \right) Y_t \right) \vec{\alpha_t} \right) \qquad (5.2) \\ \text{subject to} \quad & M \succeq 0, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, \sum_{j=1}^{J} M_{0,j} \leq 1, M_{0,0} = 1, \ \forall i \\ & \vec{0} \leq \vec{\alpha}_t \leq C\vec{1}, \vec{\alpha_t}^T \vec{y} = 0, \ \forall t. \end{aligned}$$

This saddle point optimization can be converted into a semidefinite program similar to the single task case by first finding the dual with respect to $\{\vec{\alpha}_1, \ldots, \vec{\alpha}_T\}$ and then using the Schur complement lemma to arrive at:

$$\begin{aligned} \min_{M, s_{1,..,T}, \lambda_{1,..,T}, \vec{\nu}_{1,..,T}, \vec{\delta}_{1,..,T}} \quad & \sum_t s_t \qquad (5.3) \\ \text{subject to} \quad & \begin{pmatrix} Y_t \sum_{i,j} M_{i,j} \phi_i(X_t)^T \phi_j(X_t) Y_t & \vec{1} + \vec{\nu}_t - \vec{\delta}_t + \lambda_t \vec{y}_t \\ (\vec{1} + \vec{\nu}_t - \vec{\delta}_t + \lambda_t \vec{y}_t)^T & t - 2C\vec{\delta}_t^T \vec{1} \end{pmatrix} \succeq 0, \forall t \\ & M \succeq 0, \sum_{j=1}^{J} M_{j,j} \leq 1, M_{0,i} \geq 0, \sum_{j=1}^{J} M_{0,j} \leq 1, M_{0,0} = 1, \forall i \\ & \vec{\nu}_t \geq \vec{0}, \vec{\delta}_t \geq \vec{0}, \forall t. \qquad (5.4) \end{aligned}$$

Unfortunately, this SDP has $T+1$ semidefinite blocks which will yield a very slow algorithm.

The extragradient algorithm will again provide an alternative algorithm to solve this relaxed problem. It can find the solution to the saddle point problem in (5.2), thus avoiding the extra T semidefinite constraints that arise from transforming it into an SDP.

Both the projections and gradients needed for the multitask extragradient algorithm are

very similar to the single task case. The needed projections are:

$$\begin{aligned} P_M(M) = \min_{\hat{M}} \quad & \|\hat{M} - M\|_F^2 \\ \text{subject to:} \quad & \hat{M} \succeq 0, \sum_{j=1}^{J} \hat{M}_{j,j} \leq 1, \hat{M}_{0,i} \geq 0, \sum_{j=1}^{J} \hat{M}_{0,j}\vec{1} \leq 1, \hat{M}_{0,0} = 1, \ \forall i \end{aligned} \tag{5.5}$$

$$\begin{aligned} P_{\alpha_t}(\vec{\alpha}_t) = \min_{\hat{\alpha}_t} \quad & \|\hat{\alpha}_t - \vec{\alpha}_t\|_2^2 \\ \text{subject to:} \quad & \vec{0} \leq \hat{\alpha}_t \leq C\vec{1}. \end{aligned} \tag{5.6}$$

They are exactly the same as the single task setting and can be solved efficiently with Dykstra's algorithm as described in Chapter 4. The gradients for $\vec{\alpha}_t$ can be computed separately for each task, but the gradients for the joint mixing matrix M are tied across the tasks. These gradients are computed as:

$$\nabla_{M_{i,j}} S(M, \alpha) = \sum_{t=1}^{T} \vec{\alpha_t}^T \left(Y_t \left(\phi_i(X_t)^T \phi_j(X_t) \right) Y_t \right) \vec{\alpha_t} \tag{5.7}$$

$$\nabla_{\alpha_t} S(M, \alpha) = 2\vec{1} + 2 \left(Y_t \left(\sum_{i,j} M_{i,j} \phi_i(X_t)^T \phi_j(X_t) \right) Y_t \right) \vec{\alpha_t}. \tag{5.8}$$

The extragradient algorithm, listed as Algorithm 4, can efficiently solve the joint optimization over multiple related tasks without having to rely on a large semidefinite program which is not computationally feasible.

Algorithm 5 Multitask Convex Matrix Mixture of Transformation Learning

$k = 0$

while not converged **do**

repeat

Prediction:

$\bar{M}^{k+1} = P_M \left(M^k - \beta \nabla_M S(M^k, \alpha^k_{(1,\ldots,T)}) \right)$

for $t = 1$ to T **do**

$\bar{\alpha}^{k+1}_t = P_{\alpha_t} \left(\alpha^k_t + \beta \nabla_{\alpha_t} S(M^k, \alpha^k_{(1,\ldots,T)}) \right)$

end for

$r = \beta \frac{\|\nabla S(M^k, \alpha^k_{(1,\ldots,T)}) - \nabla S(\bar{M}^{k+1}, \bar{\alpha}^{k+1}_{(1,\ldots,T)})\|}{(\|M^k - \bar{M}^{k+1}\| + \|\alpha^k_{(1,\ldots,T)} - \bar{\alpha}^{k+1}_{(1,\ldots,T)}\|)}$

if $r \geq \nu$ **then**

$\beta = (2/3)\beta \min(1, 1/r)$

end if

until $r < \nu$

Correction:

$M^{k+1} = P_M \left(M^k - \beta \nabla_M S(\bar{M}^{k+1}, \bar{\alpha}^{k+1}_{(1,\ldots,T)}) \right)$

for $t = 1$ to T **do**

$\alpha^{k+1}_t = P_{\alpha_t} \left(\alpha^k_t + \beta \nabla_{\alpha_t} S(\bar{M}^{k+1}, \bar{\alpha}^{k+1}_{(1,\ldots,T)}) \right)$

end for

k=k+1

end while

Chapter 6

Learning Monotonic Transformations

6.1 Monotonic Regression

Monotonic functions are a useful tool in statistical modeling and machine learning. They can be used directly in regressions where the dependent variable has a monotonic relation to the independent variable [SS04]. For example, growth curves have this characteristic. Age is typically assumed to be monotonically related to height. Monotonic classifiers have also been studied where the feature vectors have a monotonic relationship with binary class labels [Sil98]. Monotonic functions can also serve a more utilitarian role in statistics such as in the Box Cox transform [BC64] which estimates a parameterized monotonic transformation of the dependant variable so that the overall regression is more normally distributed. Monotonic transformations of the data are a useful preprocessing step for support vector machines that improves classification accuracy in many domains [JKH04, HB05, CHV99, BGL$^+$00]. In this chapter, we will be learning these monotonic transformations automatically using the maximum margin transformation learning algorithm.

We will start by looking at the simple case of monotonic regression (also known as isotonic regression) in order to define the building blocks necessary to integrate learned monotonic transformations in the support vector machine framework. Isotonic regression is a nonparametric shape restricted regression in which the shape restriction takes the form of

order constraints. Other shape restrictions include functional properties such as convexity and unimodality [SS04]. Formally, a function is said to be monotonic if $f(x) \geq f(x')$ whenever $x \geq x'$ for any x and x'. The monotonic regression problem can be stated as: given as a set of monotonically related observations $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ find a monotonic function $f(x)$ that minimizes the sum of squared errors $\sum_i (f(x_i) - y_i)^2$. A simple class of nonparametric monotonic functions are the piecewise linear functions. In order to enforce monotonicity, the right end of each line segment is constrained to be higher (or lower) than the left side.

Isotonic regression is also appealing from a theoretical point of view. It is the maximum likelihood estimator under the assumption that the residuals have a normal distribution [Bru55]. It was shown to have strong uniform consistency on closed and bounded intervals [HPW73]. Additionally, the asymptotic distribution has been well studied. The asymptotic distribution of the L_1 error was shown to be of the magnitude of $n^{\frac{1}{3}}$ and that a centered version of this distance converges at the rate of $n^{\frac{1}{2}}$ to a Gaussian random variable with fixed variance [Dur02].

The isotonic regression with piecewise linear functions can be solved with the following quadratic program:

$$\begin{aligned} \min_{\vec{f}} \quad & \sum_{i=1}^{N} (f_i - y_i)^2 \\ \text{subject to} \quad & f_i \geq f_{i-1} \; \forall i \end{aligned} \tag{6.1}$$

where $f_i = f(x_i)$ is the function value as the observed points x_i. The x_i's are assumed to be in ascending order so that the inequality constraints enforce monotonicity. Note that the actual value of the x's are unimportant, only their ordering. Additionally, the algorithm only estimates the values of $f(x_i) = f_i$ at the observed points. Any values between these observed points can be found through linear interpolation $f(x) = \frac{f_i - f_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}) + f_{i-1}$.

This problem is solvable with standard quadratic program solvers, but a simpler method known as the pool adjacent violators (PAV) algorithm can be used instead. The PAV algorithm scans forward through the data looking for any examples that violate the monotonicity constraint. If a violator has been found, it then scans backward pooling all violators and

averaging them until no more violators are found. Pseudo code for the algorithm can be found as Algorithm 6.

Algorithm 6 Pool Adjacent Violators

$f_1 = y_1$
for $i = 2$ to N **do**
  $f_i = y_i$
  $n = 0$
  while $f_{i-n} < f_{i-1-n}$ and $n < i$ **do**
    $f_i = \ldots = f_{i-1-n} = (y_i + \ldots + y_{i-1-n})/(n+2)$
    $n = n + 1$
  end while
end for

Another way to solve the isotonic regression problem for the class of piecewise linear functions is to define $f(x)$ explicitly as a basis function expansion $f(x) = \sum_i m_i \phi(x)$. These basis functions are defined as truncated ramp functions with one per data point:

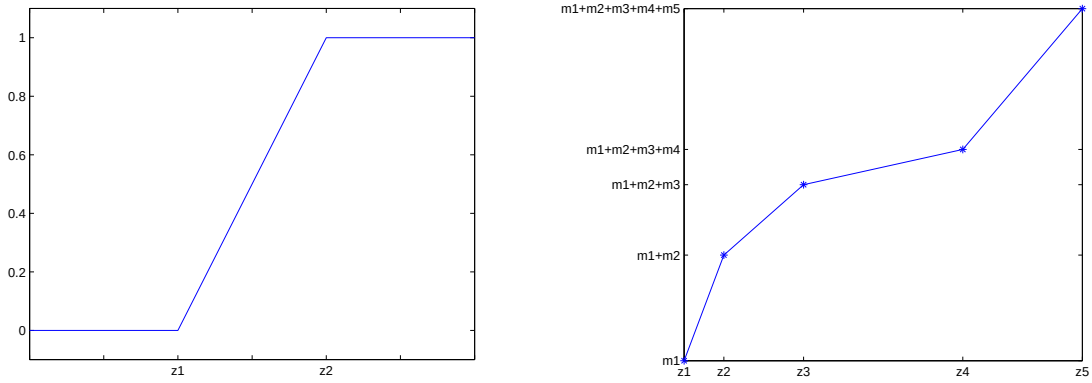
$$\phi_j(x) = \begin{cases} 0 & x \leq z_j \\ \frac{x - z_j}{z_{j+1} - z_j} & z_j < x < z_{j+1} \\ 1 & z_{j+1} \leq x. \end{cases} \tag{6.2}$$

Positivity constraints on the mixing weights, m, enforce monotonicity on $f(x)$ for all x. The function can be visualized in Figure 6.1(a) and an example of a function built with these basis functions can be seen in Figure 6.1(b).

This alternative framework also yields a quadratic program:

$$\begin{aligned} \min_{\vec{m}} \quad & \sum_{i=1}^{N} \left(y_i - \sum_j m_j \phi_j(x_i) \right)^2 \\ \text{subject to} \quad & \vec{m} \geq \vec{0} \end{aligned} \tag{6.3}$$

This QP is slightly simpler than the previous one. It has the same number of variables and constraints, but the constraints are simple positivity constraints rather than order constraints. The structure of this problem does not yield a simple algorithm such as PAV



(a) Truncated ramp function $\phi_1(x)$ (b) $f(x) = \sum_{j=1}^{5} m_j \phi_j(x)$

Figure 6.1: Building blocks for piecewise linear functions. Truncated ramp functions (a) can be combined with positive weights in the mixture of transformations framework to learn piecewise linear monotonic functions as in (b).

so, in practice, it is not the preferred method. In the next section however, this will be the preferred representation when jointly estimating a monotonic transformation and a linear hyperplane in the maximum margin transformation learning framework.

6.2 Learning Large Margin Monotonic Transformations

We now turn our attention to learning monotonic transformations of the data to improve support vector machine classification accuracy. In many domains such as document classification [JKH04, HB05], image histogram classification [CHV99] and gene microarray experiments [BGL+00], fixed monotonic transformations can be useful as a preprocessing step. However, most classifiers only explore these transformations through manual trial and error or via prior domain knowledge. The text classification problem typically uses a square root of the word frequencies, image histogram classification cross validates over transformations of the form x^a with $a \in [0, 1]$ and microarray experiments use the logarithm of expression ratios. We propose an algorithm that automatically learns monotonic transformations while training a large-margin classifier without any prior knowledge of the domain. This can be seen as a specific instance of maximum margin transformation learning with trun-

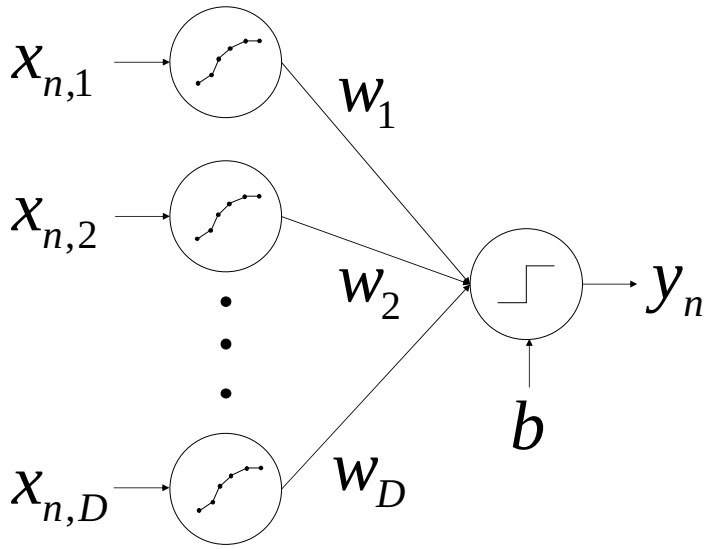


Figure 6.2: The network model shows the learned monotonic transformation first applied to each dimension of the data which is then followed by a hyperplane classifier to predict the class label. The decision rule is $y_n = \text{sign}\left(\langle \vec{w}, \sum_i m_i \phi_i(\vec{x}_n) \rangle\right)$.

cated ramp functions as base transformations. The combination of hyperplane classifier and learned transformation can be visualized in Figure 6.2.

In the previous section, it was concluded that the traditional isotonic regression method of pooled adjacent violators was preferred for its algorithmic simplicity. This defined piecewise linear monotonic function by learning values for the endpoints of each line segment as opposed to finding a conic combination of truncated ramp functions. For the more complicated task of learning a monotonic transformation and hyperplane classifier simultaneously, no analogous simple algorithm presents itself. Therefore it is important to verify that the truncated ramp function method, which corresponds to the large margin transformation learning framework, is the preferred technique.

Although, the two representations are equivalent, mixtures of truncated ramp functions are better suited to the combined hyperplane and transformation estimation. The mixture of transformations formulation yields a simpler and more elegant formulation. To verify this, we first look at the optimization with order constraints:

$$\begin{aligned} \min_{\vec{w},\vec{\xi},b,\vec{f}} \quad & \|\vec{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i\left(\sum_d w_d\left(\frac{f_{u(i,d)}-f_{l(i,d)}}{z_{u(i,d)}-z_{l(i,d)}}(x_{i,d}-z_{l(i,d)})+f_{l(i,d)}\right)+b\right)\geq 1-\xi_i\,\forall i \\ & \xi_i\geq 0, f_i\geq f_{i-1}\forall i \end{aligned} \tag{6.4}$$

where the variable $f_j = f(z_j)$ is the function value at a small number preselected knots $\{z_1,\ldots,z_J\}$. Because there are less knots then data points, the linear interpolation for a training point $\vec{x}_i$ must computed as $\frac{f_{u(i,d)}-f_{l(i,d)}}{z_{u(i,d)}-z_{l(i,d)}}(x_{i,d}-z_{l(i,d)})+f_{l(i,d)}$ where $x_{i,d}$ is the d'th dimension of $\vec{x}_i$ and the indexing functions $l(i,d)$ and $u(i,d)$ choose the index for the knot below and above $x_{i,d}$. Clearly this is an unwieldy optimization and subsequent convex relaxations yield equally complicated semidefinite programs.

Using truncated ramp functions is preferable for a number of reasons. The initial optimization problem is much simpler and is exactly an instance of large margin transformation learning:

$$\begin{aligned} \min_{\vec{w},\vec{\xi},b,\vec{m}} \quad & \|\vec{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i\left(\left\langle \vec{w}, \sum_{j=1}^{J} m_j\phi_j(\vec{x_i})\right\rangle + b\right)\geq 1-\xi_i\,\forall i \\ & \xi_i\geq 0, m_j\geq 0, \sum_j m_j\leq 1\,\forall i,j. \end{aligned} \tag{6.5}$$

As opposed to the previous method, there is no complicated indexing scheme and no need to explicitly compute linear interpolations. The positivity constraints on the weights, $\vec{m}$, yield a simpler and more elegant formulation than order constraints and interpolation which is important when deriving and implementing a relaxed algorithm. Additionally, the base transformations, $\phi(x)$, can be easily precomputed and yield a sparse representation. Once precomputed, most calculations can be done via sparse matrix multiplications. This is quite important when computing kernels on the fly in the extragradient algorithm when memory is a concern.

An important question that needs to be answered when defining truncated ramp functions is how to choose the knot locations. There are two related methods for choosing

the position of the knots. The first is to normalize the data so that $\vec{x} \in [0,1]^n$ via $\vec{x} = \frac{x - \min_{i,d}(\vec{x}_i^d)}{\max_{i,d}(\vec{x}_i^d) - \min_{i,d}(\vec{x}_i^d)}$. Once normalized, z_k is placed at the empirical quantiles ensuring that the function has expressive power where the data lies. This is explained more explicitly in Algorithm 6. The second method is to preprocess the data by mapping it through its empirical cumulative distribution function and space z_k uniformly. Algorithm 7 describes this method in more detail. Learning monotonic transformations with these knot selection schemes will yield identical learned functions in the limit of infinite data and yields similar results in practice. The methods both have the important property of placing knots with an equal number of data points between them.

Algorithm 7 Knot Choice at Quantiles

Map to [0,1] box by scaling:

for $i = 1$ to N **do**

$\hat{x}_i = \frac{x_i - \min_j(x_j)}{\max_j(x_j) - \min_j(x_j)}$

end for

Compute CDF: $F_N(x) = \frac{1}{N} \sum_{i=1}^{N} I(\hat{x}_i < x)$

Choose knots at the quantiles:

for $j = 1$ to J **do**

$z_j = x$ such that $F_N(x) = \frac{j}{J}$

end for

Algorithm 8 Uniform Knot Choice with CDF

Compute CDF: $F_N(x) = \frac{1}{N} \sum_{i=1}^{N} I(x_i < x)$

Map to [0,1] box with CDF:

for $i = 1$ to N **do**

$\hat{x}_i = F_N(x_i)$

end for

Choose knots uniformly:

for $j = 1$ to J **do**

$z_j = \frac{j}{J}$

end for

Chapter 7

Learning Transformations Via Alignment

7.1 Kernel Alignment

This chapter looks at learning transformations via kernel alignment. Previously chapters have explored a large margin approach to transformation learning, and we now look at an alternative criterion for finding optimal transformations. The (empirical) alignment between two kernel matrices is defined as:

$$A(K_1, K_2) = \frac{\langle K_1, K_2 \rangle_F}{\|K_1\|_F \|K_2\|_F}. \tag{7.1}$$

This can be seen as similarity score based on the cosine of the angle between K_1 and K_2 in an appropriate space. For arbitrary matrices, this score ranges between -1 and 1. However, since the kernel alignment is only measured on positive semidefinite Gram matrices, the score is lower bounded by 0.

The alignment gives a measure of the quality of a kernel by comparing it to an idealized kernel made up of the outer product of observed labels $\vec{y}\vec{y}^T$.

$$A(K, \vec{y}\vec{y}^T) = \frac{\langle K, \vec{y}\vec{y}^T \rangle_F}{\|K\|_F \|\vec{y}\vec{y}^T\|_F} = \frac{\langle K, \vec{y}\vec{y}^T \rangle_F}{N\|K\|_F} \tag{7.2}$$

This score defines a useful quantity to optimize when searching for an optimal kernel:

$$\begin{aligned} \max_{K} \quad & A(K, \vec{y}\vec{y}^T) \\ \text{subject to} \quad & K \in \mathcal{K}. \end{aligned} \tag{7.3}$$

Instead of maximizing the normalized score in (7.3), an alternative equivalent optimization can be derived that can be solved with standard solvers. First, the normalization is moved from the cost function into the constraints:

$$\begin{aligned} \max_{K} \quad & \left\langle K, \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \|K\|_F = 1 \\ & K \in \mathcal{K}. \end{aligned} \tag{7.4}$$

Then, noting that $\|K\|_F = 1 \iff \langle K, K \rangle_F = 1$, the norm constraint can be replaced with a quadratic inner product constraint. Additionally, the equality constraint can be changed to an inequality constraint, because (7.4) is a maximization problem:

$$\begin{aligned} \max_{K} \quad & \left\langle K, \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \langle K, K \rangle_F \leq 1 \\ & K \in \mathcal{K}. \end{aligned} \tag{7.5}$$

The inequality constraint $\langle K, K \rangle_F \leq 1$ can be transformed into a semidefinite constraint by first introducing a matrix A with bounded trace:

$$\langle K, K \rangle_F \leq 1 \tag{7.6}$$

$$\Rightarrow \quad \text{trace}(K^T K) \leq \text{trace}(A),\ \text{trace}(A) \leq 1 \tag{7.7}$$

$$\Rightarrow \quad A - K^T K \succeq 0,\ \text{trace}(A) \leq 1 \tag{7.8}$$

$$\Rightarrow \quad \begin{pmatrix} A & K^T \\ K & I_N \end{pmatrix} \succeq 0,\ \text{trace}(A) \leq 1 \tag{7.9}$$

where the last line is due to the Schur complement lemma. Substituting this result into (7.5) yields a general optimization for maximizing the alignment:

$$\begin{aligned} \max_{K} \quad & \left\langle K, \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \text{trace}(A) \leq 1 \\ & \begin{pmatrix} A & K^T \\ K & I_N \end{pmatrix} \succeq 0 \\ & K \in \mathcal{K}. \end{aligned} \tag{7.10}$$

If $\mathcal{K}$ is the set of all positive definite matrices, this optimization yields a trivial solution of $K = \frac{c}{n}\vec{y}\vec{y}^T$. This indicates that it is important to constrain K to be less general. The multiple kernel learning framework defines $K = \sum_i m_i K_i$ as a linear combination of base kernels which defines a semidefinite program:

$$\begin{aligned} \max_{m} \quad & \left\langle \sum_j m_j K_j, \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \text{trace}(A) \leq 1 \\ & \begin{pmatrix} A & K^T \\ K & I_N \end{pmatrix} \succeq 0 \\ & \sum_j m_j K_j \succeq 0. \end{aligned} \tag{7.11}$$

If the weights m_i are constrained to be positive, this defines a QCQP:

$$\begin{aligned} \max_{\vec{m}} \quad & \vec{m}^T q \\ \text{subject to} \quad & \vec{m}^T S \vec{m} \leq 1 \\ & \vec{m} \geq \vec{0} \end{aligned} \tag{7.12}$$

where $q_i = \langle K_i, \vec{y}\vec{y}^T \rangle_F$ and $S_{i,j} = \langle K_i, K_j \rangle_F$.

7.2 Learning Transformations Via Alignment

We next look at learning a mixture of transformations by maximizing the alignment score to the labels where the kernel is defined as $K = \sum_{i,j} m_i m_j \phi_i(x)^T \phi_j(x)$ with $m_i \geq 0$. This can be phrased as a similar optimization to (7.3) but with transformations:

$$\begin{aligned} \max_{m} \quad & A\left(\sum_{i,j} m_i m_j \phi_i(x)^T \phi_j(x), \vec{y}\vec{y}^T\right) \\ \text{subject to} \quad & \vec{m} \geq \vec{0} \end{aligned} \tag{7.13}$$

We can now apply the general alignment optimization derived in (7.10) to this task of learning a mixture of transformations. Defining $K_{i,j} = K_{j,i}^T = \phi_i(X)^T \phi_j(X)$ and $Q_{i,j} = Q_{j,i} = \left\langle K_{i,j}, yy^T \right\rangle_F$ we arrive at:

$$\begin{aligned} \max_{m} \quad & \vec{m}^T Q \vec{m} \\ \text{subject to} \quad & \text{trace}(A) \leq 1 \\ & \begin{pmatrix} A & \sum_{i,j} m_i m_j K_{j,i} \\ \sum_{i,j} m_i m_j K_{i,j} & I_N \end{pmatrix} \succeq 0 \\ & \vec{m} \geq \vec{0} \end{aligned} \tag{7.14}$$

Unfortunately, this optimization is nonconvex but can be relaxed using the recipe from Chapter 3:

$$\begin{aligned} \max_{M} \quad & \text{trace}(MQ) \\ \text{subject to} \quad & \text{trace}(A) \leq 1 \\ & \begin{pmatrix} A & \sum_{i,j} M_{j,i} K_{j,i} \\ \sum_{i,j} M_{i,j} K_{i,j} & I_N \end{pmatrix} \succeq 0 \\ & M \succeq \vec{0} \end{aligned} \tag{7.15}$$

This relaxation is a semidefinite program which can be solved with standard solvers. The learned kernel is $K = \sum_{i,j} M_{i,j} \phi_i(x) \phi_j(x)$ which is the matrix mixture of transformations. We call this the norm constrained transformation alignment algorithm.

7.3 Mixing Weight Constrained Transformation Alignment

Optimizing the kernel alignment $A(K, \vec{y}\vec{y}^T)$ in (7.3) can be seen as maximizing a similarity score $\langle K, \vec{y}\vec{y}^T \rangle_F$ subject to a complexity constraint based on the norm $\|K\|_F = 1$.

Other types of complexity control could be used in this setting. We now investigate simple constraints on the mixing weights of the kernel $\vec{m} \geq \vec{0}$ and $\vec{m}^T\vec{1} \leq 1$:

$$\begin{aligned} \max_{\vec{m}} \quad & \left\langle \sum_{i,j} m_i m_j \phi_i(X)^T \phi_j(X), \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \vec{m} \geq \vec{0}, \vec{m}^T\vec{1} \leq 1, \sum_i m_i^2 \leq 1, \forall i. \end{aligned} \tag{7.16}$$

This includes an additional redundant constrain $\sum_i m_i^2 \leq 1$ that will be useful in the relaxation. Introducing the positive semidefinite matrix M similar to the previous subsection yields a convex relaxation which we call the mixing weight constrained transformation alignment algorithm:

$$\begin{aligned} \max_{m} \quad & \left\langle \sum_{i,j} M_{i,j} \phi_i(X)^T \phi_j(X), \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \begin{pmatrix} 1 & \vec{m}^T \\ \vec{m} & M \end{pmatrix} \succeq 0 \\ & \vec{m} \geq \vec{0}, \vec{m}^T\vec{1} \leq 1, \sum_i M_{i,i} \leq 1 \end{aligned} \tag{7.17}$$

This method has better space and time complexity than either the norm constrained transformation alignment or large margin transformation learning. While the previous two methods have semidefinite blocks that scale with the number of data points and need to store all $N^2/2 - N$ kernel matrices in memory. This formulation requires neither and can precompute the most burdensome quantity with only matrix-vector and vector-vector multiplication:

$$\left\langle \sum_{i,j} M_{i,j} \phi_i(X)^T \phi_j(X), \vec{y}\vec{y}^T \right\rangle_F = \sum_{i,j} M_{i,j} \text{trace}\left((\vec{y}^T \phi_i(X)^T)(\phi_j(X)\vec{y}) \right). \tag{7.18}$$

7.4 Sparse Kernel and Transformation Alignment

We now investigate two alternative possibilities for controlling complexity by choosing sparse sets of kernels or transformations based on the alignment score.

We will first investigate two scenarios for learning sparse kernels and then apply these techniques to sparse transformation learning. The first method is to replace the norm constraint $\|K\|_F = 1$ with a cardinality constraint:

$$\begin{aligned} \max_{m} \quad & \left\langle \sum_i m_i K_i, \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \sum_i m_i = k \\ & m \in \{0,1\}. \end{aligned} \tag{7.19}$$

This method chooses the k best kernels and gives them even weight. The optimization has a trivial solution of simply computing each kernel's score and choosing the k highest.

Another approach is to penalize the cardinality and trade off between maximizing the score and penalizing cardinality:

$$\begin{aligned} \max_{m} \quad & \left\langle \sum_i m_i K_i, \vec{y}\vec{y}^T \right\rangle_F - C \sum_i m_i \\ \text{subject to} \quad & m \in \{0,1\}. \end{aligned} \tag{7.20}$$

This optimization also can be solved in a trivial way. Because the kernel score and the cardinality are traded off linearly, C acts as a threshold. Choosing kernels where $\langle K_i, \vec{y}\vec{y}^T \rangle_F > C$ solves the problem.

Although both of these problems are not very interesting when learning a sparse mixture of kernels, learning a sparse mixture of transformations is a difficult NP-Hard combinatorial optimization problem. Approximation algorithms based on semidefinite programs will be necessary to solve them. These approximations will again yield the matrix mixture of transformations model.

The cardinality constrained mixture of transformations alignment problem will be studied first. It can be phrased as the following optimization:

$$\begin{aligned} \max_{m} \quad & \left\langle \sum_{i,j} m_i m_j \phi_i(x)^T \phi_j(x), \vec{y}\vec{y}^T \right\rangle_F \\ \text{subject to} \quad & \sum_i m_i = k \\ & m \in \{0,1\} \end{aligned} \tag{7.21}$$

This turns out to be an example of the Dense-k-Subgraph problem defined on a complete graph with a vertex for each transformation and weights $w_{ij} = \langle \phi_i(x)^T \phi_j(x), \vec{y}\vec{y}^T \rangle_F$. Dense-$k$-Subgraph finds the subset of k vertices in a graph G that has a maximum weight set of

edges between vertices within the subset. The problem is defined formally as: given an undirected graph $G = (V, E)$ with N vertices V connected by edges E with nonnegative weights w_{ij}, determine a subset $S \subset V$ with $|S| = k$ such that $\sum_{i \in S, j \in S} w_{i,j}$ is maximized. This is an NP-Hard problem and the best approximations for it are based on semidefinite programs.

Constant factor semidefinite programming approximation algorithms were pioneered by Goemans and Williamson with the much celebrated 0.8785 approximation to Max-Cut [GW95]. Many other discrete NP-hard optimizations have been solved with similar types of algorithms. The main ingredients in these algorithms is to relax a $\{-1, 1\}$ discrete quadratic optimization into an optimization on a sphere. Once a solution is found by an SDP, a random rounding scheme is used to find a feasible solution to the original problem.

Dense-k-Subgraph is similar to Max-Cut-k which is a constrained version of the Max-Cut problem requiring a partition to have exactly k elements in it. The best approximations for Dense-k-Subgraph are slightly better than k/N. For example when $k = \frac{N}{2}$ Ye and Zhang report a 0.5866 algorithm [YZ03] with the best reported approximation as 0.6221 by Halperin and Zwick [HZ02]. These two results were only studied for the bisection problem of $k = 2/N$ in the referenced papers. However, Han, Ye and Zhang, describe a way to generalize the 0.5866 approximation algorithm to arbitrary k yielding approximations superior to k/N [HYZ02] by introducing an additional node swapping step to insure the correct number of vertices in the subset. Halperin and Zwick mention that their 0.6221 algorithm can be generalized to arbitrary k by using the rounding technique of Feige and Langberg [FL01] but do not provide an analysis.

The Dense-k-Subgraph problem can be phrased as a discrete quadratic optimization problem:

$$\begin{aligned} \max_{m} \quad & \sum_{i,j} \frac{1}{2}(m_i + 1)\frac{1}{2}(m_j + 1)w_{i,j} \\ \text{subject to} \quad & \sum_{i} \frac{1}{2}(m_i + 1) = k \\ & m \in \{-1, 1\} \end{aligned} \tag{7.22}$$

An approximation to this problem proposed by Han, Ye and Zhang [HYZ02] can be found

with a three step algorithm listed as Algorithm 9. First, a semidefinite relaxation is solved:

$$\begin{aligned} \max_{M} \quad & \frac{1}{4}\sum_{i,j} w_{i,j}(1 + M_{0,i} + M_{0,j} + M_{i,j}) \\ \text{subject to} \quad & \sum_{i,j} M_{i,j} = (2k-n)^2 \\ & \sum_{i} M_{0,i} = 2k - n \\ & M_{i,i} = 1 \forall i \\ & M \succeq 0. \end{aligned} \tag{7.23}$$

Then a randomized rounding technique listed as Algorithm 10 is used to recover discrete variables in the feasible set. This is followed by a procedure to insure that the proper number of vertices are in the partition in Algorithm 11. The various parameters in the procedure vary depending on the problem and no closed form solution exists for the approximation ratio. A table of the approximation values for various settings of k/N are in Table 7.1.

Algorithm 9 Dense-k-Subgraph Approximation

Solve SDP (7.23) for M
$w^* = 0$
for $n = 1$ to N **do**
  Use Randomized Rounding (Han, Ye, Zhang) on M to recover $\vec{m} \in \{0,1\}^{J+1}$
  Set $S = \{i : m_i = m_0\}$
  Use Node Swapping to ensure $|S| = k$
  $w = \sum_{i \in S, j \in S} \frac{1}{2}(m_i + 1)\frac{1}{2}(m_j + 1)W_{i,j}$
  if $w > w^*$ **then**
    $w^* = w$
    $S^* = S$
  end if
end for

We now turn our attention to learning a mixture of transformations via kernel alignment with penalized cardinality. This problem does not fit into the class of well studied NP Hard

Algorithm 10 Randomized Rounding (Han, Ye, Zhang)

Sample $\vec{u} \sim N(0, \theta M + (1-\theta)M)$

$\vec{m} = \text{sign}(\vec{u})$

Algorithm 11 Node Swapping

if $|S| > k$ **then**

$N = |S| - k$

for $n = 1$ to N **do**

for $i = 1$ to $|S|$ **do**

$\zeta_i = \sum_{j \in S} W_{i,j}$

end for

$S = \{s \in S : s \neq \arg\min_{i \in S} \zeta(i)\}$

end for

else if $|S| < k$ **then**

Add $k - |S|$ nodes to S arbitrarily

end if

$\frac{k}{n}$	R	$\frac{k}{n}$	R	$\frac{k}{n}$	R	$\frac{k}{n}$	R
0.20	0.2008	0.32	0.3827	0.44	0.5310	0.56	0.6287
0.22	0.2320	0.34	0.4105	0.46	0.5511	0.58	0.6402
0.24	0.2631	0.36	0.4372	0.48	0.5697	0.60	0.6488
0.26	0.2942	0.38	0.4626	0.50	0.5866	0.62	0.6539
0.28	0.3245	0.40	0.4867	0.52	0.6022	0.64	0.6563
0.30	0.3541	0.42	0.5095	0.54	0.6161	0.65	0.6571

Table 7.1: Table of approximation ratios R for various fractions k/N for Dense-k-Subgraph on N vertices.

graph theoretic algorithms such as Max-Cut. However, we can use a result of Nesterov to solve this problem with a $\frac{\pi}{2} - 1 = 0.5707$ relative accuracy [Nes98]. Nesterov showed that the following general discrete optimization program can be solved with a strategy similar to that used by Goemans and Williams:

$$\begin{aligned} \max_{\vec{m}} \quad & \vec{m}^T H \vec{m} \\ \text{subject to} \quad & \vec{m} \in \{-1, 1\}^N. \end{aligned} \tag{7.24}$$

Note that this optimization does not have any linear terms. This can be rectified by adding an additional variable m_0:

$$\begin{aligned} \max_{\vec{m}} \quad & \vec{m}^T H \vec{m} + m_0 c^T \vec{m} \\ \text{subject to} \quad & \vec{m} \in \{-1, 1\}^{N+1}. \end{aligned} \tag{7.25}$$

Whenever $m_0 = 1$ then the solution $\vec{m}$ is optimal and when $m_0 = -1$ then $-\vec{m}$ is optimal. This augmentation allows for analysis involving only quadratic terms.

The penalized cardinality transformation alignment problem can be solved with the following optimization:

$$\begin{aligned} \max_{m} \quad & \left\langle \sum_{i,j} .5(m_i + 1).5(m_j + 1) K_{i,j}, \vec{y}\vec{y}^T \right\rangle_F - C \sum_i m_i \\ \text{subject to} \quad & m \in \{-1, 1\}. \end{aligned} \tag{7.26}$$

In order to transform it into problem (7.24) set:

$$H_{i,j} = \left\langle K_{i,j}, yy^T \right\rangle \tag{7.27}$$

$$H_{i,0} = H_{0,i} = \sum_j \left\langle K_{i,j}, yy^T \right\rangle - C/4. \tag{7.28}$$

This optimization can be relaxed with the normal recipe as:

$$\begin{aligned} \max_{M} \quad & \text{trace}(HM) \\ \text{subject to} \quad & \text{diag}(M) = \vec{1}, M \succeq 0. \end{aligned} \tag{7.29}$$

A feasible point $\vec{m} \in \{-1, 1\}$ can be recovered using the standard randomized rounding technique originally proposed by Goemans and Williamson [GW95]. Choose a centered

hyperplane $\vec{w}$ uniformly at random and cut the unit sphere in half assigning each side of the cut to -1 and 1 respectively. This can be seen in more detail as Algorithm 13. The entire approximation algorithm can be found as Algorithm 12. The relaxation yields a $\frac{\pi}{2} - 1 = 0.5707$ expected relative accuracy where the relative accuracy is defined as:

$$\frac{\overline{q} - q_{\text{approx}}}{\overline{q} - \underline{q}}, \tag{7.30}$$

with $\overline{q}$ being the maximum value of the cost function defined in (7.24), $\underline{q}$ being the minimum value of the cost function and q_{approx} being the value of an approximation. For comparison, the relative accuracy ranges between 0 and 1 with smaller values giving better approximations. The MAX-CUT approximation has a $1 - 0.8785$ relative accuracy.

Algorithm 12 Discrete Quadratic Optimization Approximation Algorithm

Solve SDP (7.24) for M

Use Randomized Rounding (Goemans and Williamson) on M to recover $\vec{m} \in \{0, 1\}^{J+1}$

Algorithm 13 Randomized Rounding (Goemans and Williamson)

Compute $V^T V = M$ via Cholesky decomposition

Sample $\vec{w}$ from a uniform distribution

$\vec{m} = \text{sign}(V^T \vec{w})$

Chapter 8

Experiments

8.1 Overview

In this chapter, experiments are presented to demonstrate the effectiveness of the proposed methods for learning monotonic transformations. Additionally, empirical running times of the SDP solver and the extragradient algorithm are compared as well as the running times of the alignment based methods. Finally, the multitask algorithm is shown to yield improvements by learning a common transformation among multiple related tasks. Preliminary results were presented in [HJ08b] with more extensive results in [HJ08a].

The models and algorithms of the paper are investigated in these experiments and demonstrate the need for learning transformations rather than kernels in various domains. The mixture of transformations, which is trained greedily with Algorithm 1, is reported as "Mix. Trans.". The matrix mixture of transformations is trained with either the SDP of Equation 3.13 or the extragradient algorithm from Section 4 depending on the size of the dataset and is reported as "Matrix Mix. Trans.". Both representations use the truncated ramp functions of Equation 6.2 to build monotonic transformations of the data. An obvious competitor is the mixture of kernels which is compared to the proposed methods by defining base kernels as inner products between truncated ramp functions and reported as "Mix. Kernels". The alignment based algorithms from Chapter 7 are also tested. The standard kernel alignment algorithm with truncated ramp functions reported as "Kernel Alignment" is compared to the norm constrained transformation alignment reported as "Trans. Align.

Norm" and the mixing weight constrained transformation alignment reported as "Trans. Align. Mixing". Other baseline algorithms include the linear SVM reported as "Linear", the SVM with a polynomial kernel reported as "Poly", and the SVM with a radial basis functions kernel reported as "RBF".

8.2 Synthetic Experiment

We present a simple synthetic experiment that demonstrates the methods' ability to recover a monotonic transformation from data. Points were sampled near a linear decision boundary and were labeled based on which side of the separating hyperplane they fell as in Figure 8.1(a). The data was then transformed with a strictly monotonic function such as the normalized logarithm in Figure 8.1(b) or a quadratic as in 8.1(c). The training set is made up of the transformed points and the original labels. A linear algorithm will have difficulty because the mapped data is no longer linearly separable. However, if the inverse monotonic function were recovered and applied to the data points, then a linear decision boundary would perform well.

For this experiment, 600 data points were sampled, and then transformed with the various monotonic functions. They were then split evenly into training, cross validation and testing sets. The models proposed in this thesis, the mixture of transformations (Mix. Trans.), the matrix mixture of transformations (Matrix Mix. Trans.), the norm constrained transformation alignment (Trans. Align. Norm), and the mixing weight constrained transformation alignment (Trans. Align. Mix) were compared to a linear SVM (Linear), a mixture of kernels (Mix. Kernels) trained with the large margin criteria, and a mixture of kernels found by maximizing alignment (Kernel Alignment) where the base kernels were constructed as inner products between the same truncated ramp functions used by the transformation learning algorithms.

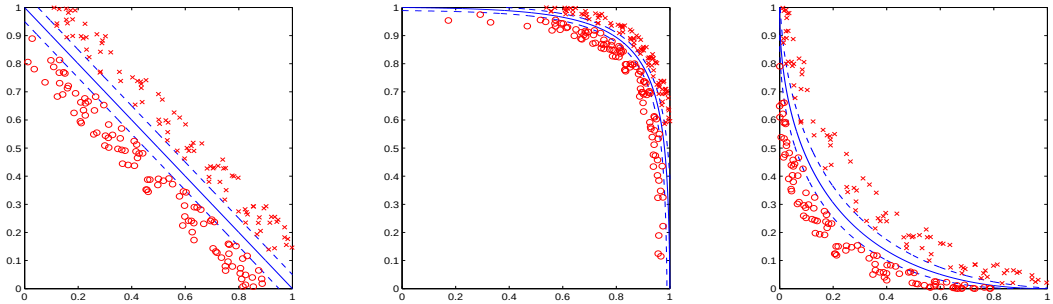
The linear SVM struggled on the transformed data and the two norm constrained alignment algorithms performed poorly. The kernel alignment and norm constrained transformation alignment end up penalizing transformations or kernels with large norm which unfairly differentiates between different ramp functions. The other methods performed quite well

as reported in Table 8.1. The p-values for a paired t-test are reported in in Table 8.2. The learned functions from the mixture of transformations are plotted in Figure 8.1(d-f). The solid blue line is the mean over 10 experiments, and the dashed blue is the standard deviation. The dashed black line is the true target function. The learned functions for the matrix mixture of transformations are in Figure 8.1(g-i) and similarly plotted in red. Both methods performed quite well on the task of classification and recovered nearly the exact monotonic transformation. The matrix mixture, learned with the convex algorithm, outperformed the greedily learned mixture of transformations by avoiding local minima. The alignment algorithm ends up outperforming the large margin matrix mixture method slightly. All three transformation learning methods outperformed the mixture of kernels algorithm because they are searching over the correct class of transformations. The mixture of kernels essentially concatenates each base transformation instead of adding them to build monotonic functions.

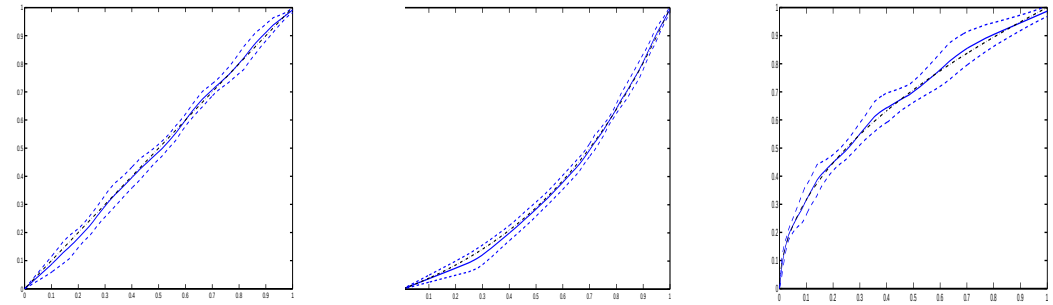
8.3 Image Histogram Classification

This experiment used the Corel image dataset which is made up of images split into various categories each containing 100 examples. Four categories of animals were chosen: (1) eagles, (2) elephants, (3) horses, and (4) tigers. Images were then transformed into RGB histograms following the binning strategy of [CHV99, HB05]. In [CHV99], it was shown that monotonic transforms of the form x^a for $0 \leq a \leq 1$ substantially outperform linear, RBF and polynomial SVMs.

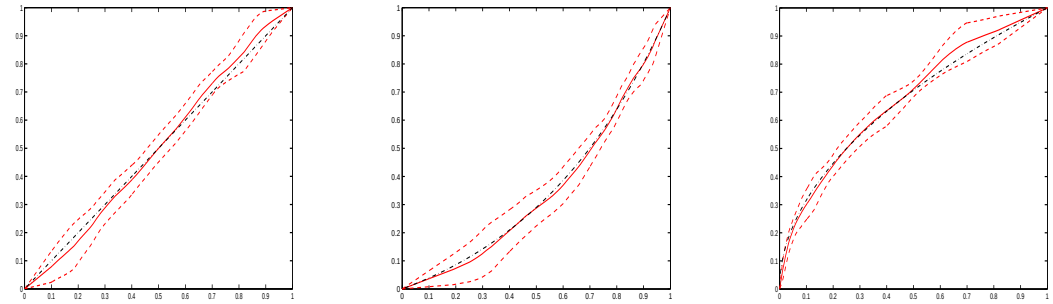
All six pair wise classification experiments where repeated 10 times with the data randomly split into 80 percent training, 10 percent cross validation, and 10 percent testing. The mixture and matrix mixture of transformations were compared to a linear support vector machine, SVMs with RBF and polynomial kernels, and a mixture of kernels based on truncated ramp functions as well as the three alignment based algorithms. Transformations of the form x^a for $0 \leq a \leq 1$ were also compared where a was chosen by cross validating over $a = \{0, .125, .25, .5, .625, .75, .875, 1\}$. This parameter set includes linear $a = 1$ at one end, a binary threshold $a = 0$ at the other (choosing $0^0 = 0$), and the square root transform



(a) $\Phi^{-1}(x) = x$ (b) $\Phi^{-1}(x) = \log(x)$ (c) $\Phi^{-1}(x) = x^2$



(d) $\Phi(x) = x$ (e) $\Phi(x) = \exp(x)$ (f) $\Phi(x) = \sqrt{x}$



(g) $\Phi(x) = x$ (h) $\Phi(x) = \exp(x)$ (i) $\Phi(x) = \sqrt{x}$

Figure 8.1: Data is sampled near a linear decision surface with margin in (a). A logarithmic transform is applied to the data and hyperplane in (b) and the square root transform is applied in (c). Then, the inverse transform and hyperplane are learned simultaneously from the transformed data. The learned inverse transforms are shown in the remaining plots. The target transform is plotted in dashed black. The average learned function from the mixture of transformations are in solid blue with the standard deviation in dashed blue (d)-(f). The average learned transformation from the matrix mixture of transformations is in solid red with the standard deviation in dashed red (g)-(i).

	Linear	**Exponential**	**Square Root**	**Avg.**
Linear	**0.00**	3.00	4.75	2.58
Mix. Kernels	1.15	1.25	1.00	1.13
Mix. Trans.	0.40	0.35	0.55	0.43
Matrix Mix. Trans.	0.20	0.20	**0.35**	0.25
Kernel Alignment	12.55	11.95	12.20	12.23
Trans. Align. Norm	6.75	6.30	8.65	7.23
Trans. Align. Mix	0.15	**0.05**	**0.35**	**0.18**

Table 8.1: Percent testing error rates for the synthetic experiments. Data was sampled near a linear decision surface then transformed with a monotonic function. The inverse of the transformation must be learned in order to predict well. The mixture of transformations (Mix. Trans.) and matrix mixture of transformations (Matrix Mix. Trans.) were compared to a linear SVM (Linear), and a mixture of kernels (Mix. Kernels). The alignment based algorithms were also compared, (Kernel Alignment), (Trans. Align. Norm) and (Trans. Align. Mix). Results for each dataset for the three target monotonic function, linear, exponential and square root, are reported in their respective columns as well as the average in the Avg. column. The results are averaged over 10 runs and the best error rate for each experiment is in bold.

	Linear	**Exponential**	**Square Root**	**Avg.**
Linear	-	0.0001	0.0003	0.0000
Mix. Kernels	0.0432	0.0735	0.1636	0.1023
Mix. Trans.	0.0368	0.0811	0.5086	0.1472
Matrix Mix. Trans.	0.0386	0.1934	-	0.5338
Kernel Alignment	0.0000	0.0000	0.0000	0.0000
Trans. Align. Norm	0.0000	0.0000	0.0000	0.0000
Trans. Align. Mix	0.0811	-	-	-

Table 8.2: P-values for the synthetic experiment. A paired t-test was conducted which compared the best result for each column to the other algorithms in the column. A "-" signifies the top result or tie for top result.

in the middle.

The matrix mixture learned with the large margin algorithm and the one learned via alignment performed best or tied for best on 4 out of 6 of the experiments and were the top two performers overall as reported in Table 8.3. The p-values for the associated paired t-tests are found in Table 8.4. The mixture of transformations also performed well tying the kernel mixture on the average error rate. All three methods based on large margin transformation or kernel learning and the mixing weight constrained alignment algorithm outperformed the cross validated family of x^a transforms. A key insight learned about this dataset is that most of the data is very close to zero due to few pixels being in a given bin so that the first few pixels are very important. Cross validation over x^a most often chose low nonzero a values which corresponds to transformations that quickly spike and then saturate. The greedy mixture of transformations method and convex matrix mixture of transformations learned similar types of monotonic functions. Plots of the learned functions can be found in Figure 8.2(a-f). The mean functions learned with the mixture of transformations are plotted in solid blue and the mean function learned with the matrix mixture of transformations is plotted in dashed red. The matrix mixture tended to find smoother functions, because it did not get stuck in local minima.

Figure 8.3 shows the effects of the learned transformations on the contribution of each color to the decision rule. Figure 8.3(a) shows the original images for an example from the tiger class and the horse class. Figure 8.3(b) shows, $S_{SVM}(c)$, the color score images for SVM classifier and 8.3(c) shows, $S_{MT}(c)$, the color score image for the learned mixture of transformations classifier. The color score is a measure of how much an individual color contributes to the decision rule. The color score for a color c is defined as $S(c) = \left(w_{b(c)}\Phi\left(x_{b(c)}\right)\right)$ where $b(c)$ is the histogram bin associated with color c. The color score can be used to compute a labeling of an image as $y = \text{sign}\left(\sum_c S(c)/Z + b\right)$ where Z is the number of colors assigned to a single bin. This score is a measure of how important a given color is to the classification rule. When the color score is multiplied by the label, $yS(c)$, white corresponds to the highest score and black the lowest. The mixture of transformations assigned the highest score to the pixels that make up the tiger and the horses which indicates that the learned decision rule is focusing areas of interest in the image. In contrast, the

	1 vs 2	**1 vs 3**	**1 vs 4**	**2 vs 3**	**2 vs 4**	**3 vs 4**	**Avg.**
Linear	10.0	6.0	2.0	13.0	7.5	10.0	8.08
Poly	9.5	6.0	2.0	12.5	7.0	9.0	7.67
RBF	7.0	6.0	2.0	9.0	7.0	8.5	6.58
$\boldsymbol{x^a}$	3.0	**0.5**	2.5	1.5	3.0	5.0	2.58
Mix. Kernels	4.5	1.0	1.5	**0.5**	3.5	3.5	2.42
Mix. Trans.	3.5	1.0	2.0	1.0	4.5	2.5	2.42
Matrix Mix. Trans.	**2.5**	1.0	**0.5**	1.0	2.0	**1.0**	1.33
Kernel Align.	3.5	1.0	3.5	3.5	8.5	6.5	4.42
Trans. Align. Norm	3.0	2.0	2.0	4.0	5.0	7.5	3.92
Trans. Align. Mix	**2.5**	1.0	**0.5**	1.0	**1.5**	**1.0**	**1.25**

Table 8.3: Percent testing error rates on Corel image histogram dataset. Four classes of animals, (1) eagles, (2) elephants, (3) horses, and (4) tigers were used in all 6 pairwise classification tasks. The average error rate is reported in the Avg. column. The mixture of transformations (Mix. Trans.) and matrix mixture of transformations (Matrix Mix. Trans.) were compared to a linear SVM (Linear), SVMs with polynomial (Poly) and RBF (RBF) kernels, an SVM with cross validated transforms (x^a) and a mixture of kernels (Mix. Kernels). The alignment based algorithms were also compared, (Kernel Alignment), (Trans. Align. Norm) and (Trans. Align. Mix). Results are averaged over 10 runs and the best error rate for each experiment is in bold.

linear SVM mistakenly gave pixels in the background the highest score which makes it difficult to distinguish between classes.

8.4 Document Classification

This experiment used the WebKB four universities dataset which consists of webpages split into multiple categories. The dataset consists of 1641 student webpages, 1124 faculty webpages, 930 course webpages, and 540 project webpages. These pages were processed to remove the HTML tags and then converted into the bag of words representation so that

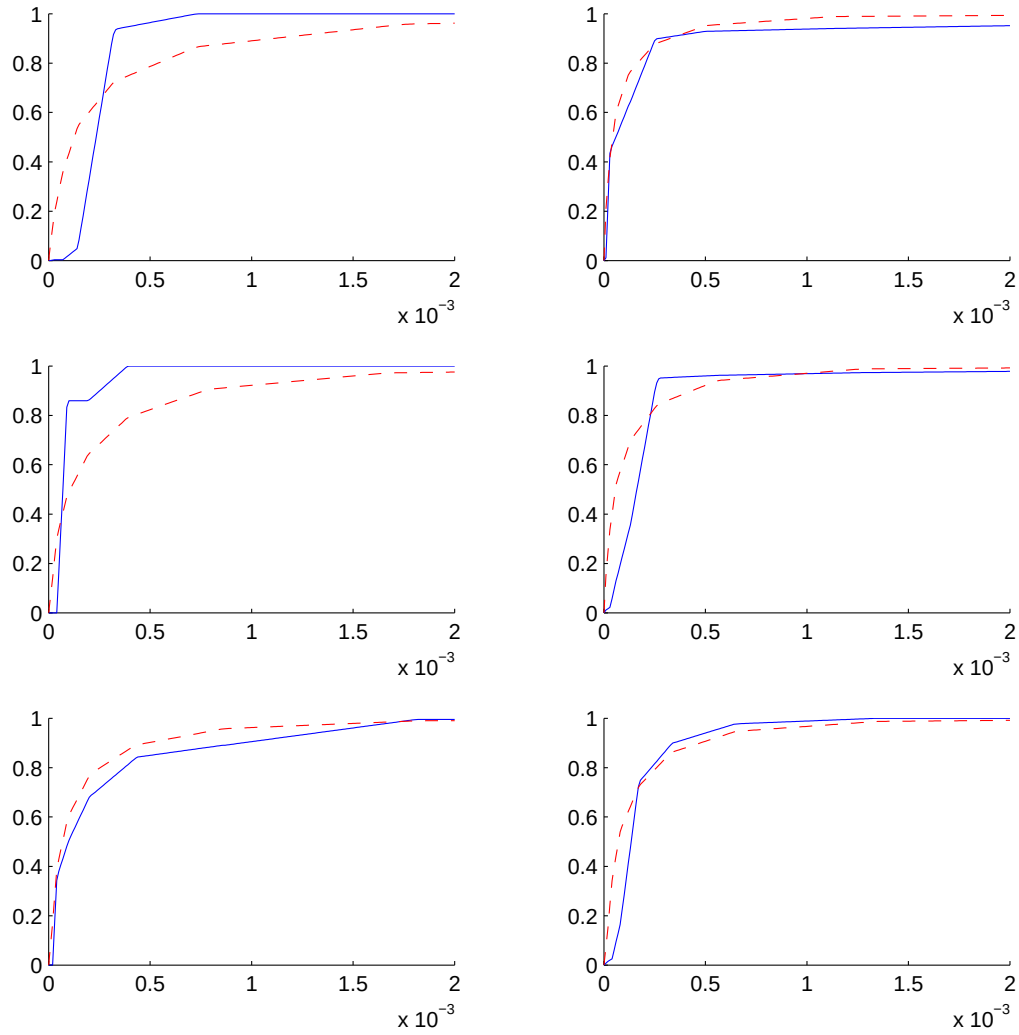
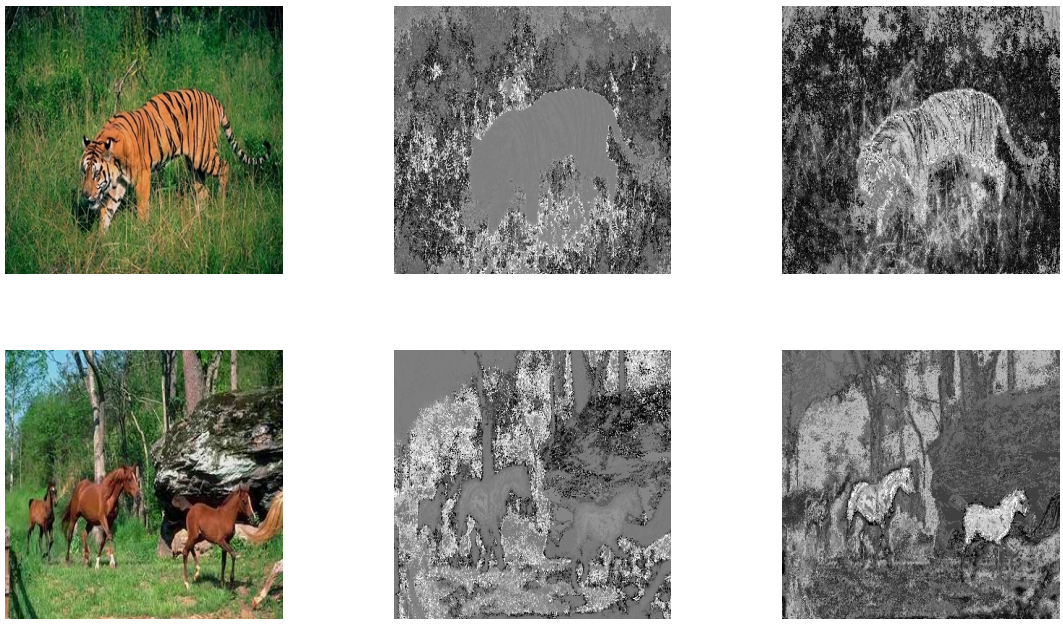


Figure 8.2: The learned transformation functions for all six Corel image histogram classification problems. The functions learned with a mixture of transformations are plotted in solid blue. The functions learned with the matrix mixture are transformations is plotted in dashed red.



(a) Original images. (b) $yS_{SVM}(c)$. (c) $yS_{MT}(c)$.

Figure 8.3: Color score images for some examples from the Corel image histogram data set. The original images of a tiger and horses are shown in (a). Models were learned for the tigers vs horses binary classification problem. The color score for a color c is computed from the learned hyperplane and monotonically transformed histogram data: $S(c) = \left(w_{b(c)}\Phi\left(x_{b(c)}\right)\right)$ where $b(c)$ is the histogram bin associated with color c. The color score can be used to compute a labeling of an image as $y = \text{sign}\left(\sum_c S(c)/Z + b\right)$ where Z is the number of colors assigned to a single bin. This score is a measure of how important a given color is to the classification rule. When the color score is multiplied by the label, $yS(c)$, white corresponds to the highest score and black the lowest. The color score images based on the linear SVM, with $\Phi(x) = x$ fixed, are shown in (b). The images based on the mixture of transformations color score, with $\Phi(x) = \sum_i m_i\phi_i(x)$ learned from data, are shown in (c). The mixture of transformations assigned the highest score to pixels in both the tiger and horses which indicates that the learned decision rules locked onto important discriminative features in the image. However, the SVM mistakenly gave pixels in the background the highest score which made it difficult to distinguish between the classes.

	1 vs 2	**1 vs 3**	**1 vs 4**	**2 vs 3**	**2 vs 4**	**3 vs 4**	**Avg.**
Linear	0.1197	0.0067	0.0811	0.0004	0.0010	0.0002	0.0001
Poly	0.1376	0.0067	0.0811	0.0003	0.0032	0.0031	0.0001
RBF	0.2531	0.0067	0.0811	0.0058	0.0032	0.0030	0.0003
x^a	0.7263	-	0.1039	0.1679	0.1934	0.1825	0.0020
Mix. Kernels	0.1676	0.3434	0.4433	-	0.1679	0.0957	0.0830
Mix. Trans.	0.3434	0.5911	0.3434	0.3434	0.0239	0.1934	0.0295
Matrix Mix. Trans.	-	0.5911	-	0.3434	0.3434	-	0.3434
Kernel Align.	0.6193	0.3434	0.1114	0.0811	0.0013	0.0243	0.0143
Trans. Align. Norm	0.7577	0.1939	0.0811	0.0248	0.0662	0.0063	0.0000
Trans. Align. Mix	-	0.5911	-	0.3434	-	-	-

Table 8.4: P-values for the Corel image histogram experiment. A paired t-test was conducted which compared the best result for each column to the other algorithms in the column. A "-" signifies the top result or tie for top result.

each page was represented by a vector of word frequencies. All six pairwise classification experiments where repeated 10 times with the data randomly split into 80 percent training, 10 percent cross validation, and 10 percent testing.

In [JKH04, HB05], preprocessing the bag of words representation by applying the square root was shown to yield good results. The matrix mixture and mixture of transformations models learned with the large margin criteria as well as the alignment based algorithms were compared to a linear SVM with and without this square root transformation, kernelized SVMs with either the polynomial kernel or the RBF kernel and a mixture of kernels based on truncated ramp functions. The linear support vector machine and both the polynomial and RBF kernels are known to perform worse than a simple square root transformation leading to the intuition that it is important to properly limit the effects of large word counts. The kernel alignment algorithm and the norm constrained transformation alignment algorithm were not compared due to the memory requirements of storing all of the kernels.

The large margin matrix mixture and the alignment algorithm with constrained mixing weights each performed best in 2 out of 6 of the experiments with the large margin version

tied for the best overall error rate with the square root transform as shown in Table 8.5 and the p-value for the associated t-tests are found in Table 8.6. The greedy mixture of transformations outperformed the kernel mixture with the alignment algorithm performing slightly worse overall. All of these transformation learning algorithms outperformed the linear and kernel based models. This demonstrates that not only is it important to use monotonic transformations but that it is possible to learn them from data given the proper algorithm and representation.

The learned functions can be visualized in Figure 8.4 with the matrix mixture of transformation plotted in dashed red and the mixture of transformations plotted in solid blue. The matrix mixture of transformations tends to learn smoother functions similar to previous experiment in Section 8.3. However, the learned functions are quite similar between the two methods, with the main difference being that the learned functions from the matrix mixture increase more quickly.

8.5 Gender classification

This experiment consists of images of males and females with the task of differentiating gender. There are 1755 labeled images from the FERET dataset which have been processed as in [MY00]. Each processed image is a 21 by 12 pixel 256 color gray scale image that is rastorized to form data vectors. The dataset consists of 1044 male images and 711 female images which were randomly split into 80 percent training, 10 percent cross validation, and 10 percent testing. The three transformation learning algorithms were compared to a linear SVM and a mixture of kernels on ten random splits of the data.

The convex matrix mixture of transformations algorithm outperforms the linear SVM and is comparable to the mixture of kernels as shown in Table 8.7. Unfortunately, in this experiment, the greedy mixture of transformations and the alignment algorithm does not seem to help performance. This is perhaps an example where the matrix mixture is useful because it is more flexible than the mixture of transformations and can explore a superset of the class of functions that the kernel mixture defines. It also seems that the large margin criteria is preferred to the alignment criteria in this problem. The learned monotonic

	1 vs 2	**1 vs 3**	**2 vs 3**	**1 vs 4**	**2 vs 4**	**3 vs 4**	**Avg.**
Linear	2.73	2.59	6.30	2.06	6.67	3.46	3.97
Sqrt	2.15	1.89	3.77	1.48	4.60	**2.29**	**2.69**
TFIDF	2.63	2.03	6.48	1.40	6.92	2.43	3.65
Poly	2.78	2.45	6.42	2.18	6.63	3.08	3.92
RBF	2.88	2.52	5.99	2.26	6.52	2.99	3.86
Mix. Kernels	1.56	2.17	**3.27**	1.95	5.76	2.52	2.87
Mix. Trans.	1.76	2.31	4.01	1.36	4.67	2.66	2.80
Matrix Mix. Trans.	**1.32**	2.03	4.51	1.13	**4.35**	2.80	**2.69**
Trans. Align. Mix	1.80	**1.82**	4.44	**1.09**	5.51	2.71	2.90

Table 8.5: Percent testing error rates for the WebKB bag of words webpage classification. Four classes of webpages, (1) student, (2) faculty, (3) course, and (4) projects were used in all 6 pairwise classification tasks. The average error rate is reported in the Avg. column. The mixture of transformations (Mix. Trans.), matrix mixture of transformations (Matrix Mix. Trans.) and the alignment algorithm (Trans. Align. Mix) were compared to a linear SVM (Linear), an SVM with a square root preprocessing (Sqrt), an SVM with text frequency inverse document frequency features (TFIDF), SVMs with polynomial (Poly) and RBF (RBF) kernels and a mixture of kernels (Mix. Kernels). Results are averaged over 10 runs and the best error rate for each experiment is in bold.

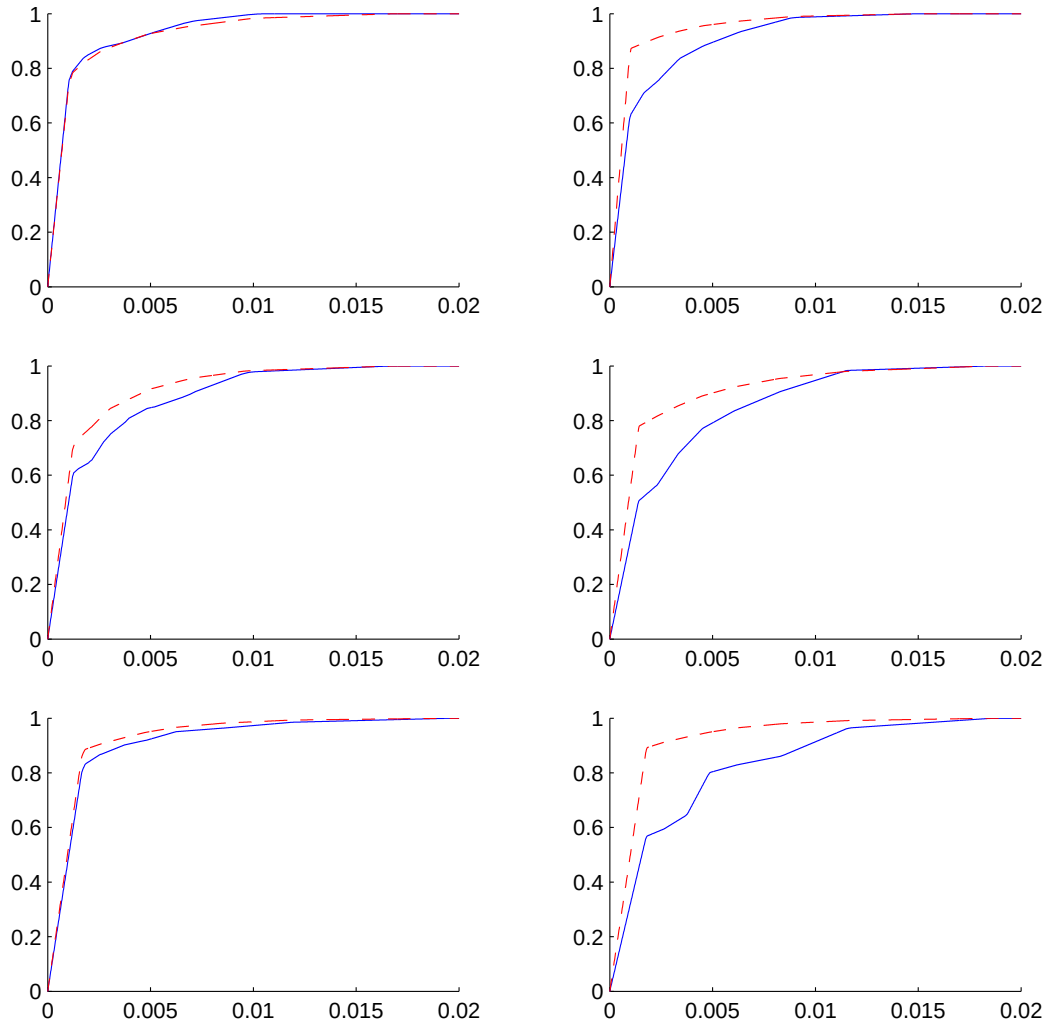


Figure 8.4: The learned transformation functions for all six WebKB text classification problems. The functions learned with a mixture of transformations are plotted in solid blue. The functions learned with the matrix mixture are transformations is plotted in dashed red.

	1 vs 2	**1 vs 3**	**2 vs 3**	**1 vs 4**	**2 vs 4**	**3 vs 4**	**Avg.**
Linear	0.0010	0.0932	0.0002	0.0127	0.0001	0.0358	0.0000
Sqrt	0.0020	0.8402	0.3270	0.0319	0.2091	-	-
TFIDF	0.0006	0.3434	0.0017	0.2100	0.0002	0.6849	0.0003
Poly	0.0006	0.1341	0.0002	0.0005	0.0000	0.0192	0.0000
RBF	0.0003	0.1173	0.0008	0.0006	0.0006	0.0150	0.0000
Mix. Kernels	0.4951	0.5960	-	0.0571	0.0477	0.6783	0.2082
Mix. Trans.	0.0100	0.0662	0.1188	0.0886	0.3729	0.3270	0.2837
Matrix Mix. Trans.	-	0.3938	0.0875	0.7976	-	0.0842	-
Trans. Align. Mix	0.0418	-	0.0521	-	0.0003	0.1708	0.0220

Table 8.6: P-values for the WebKB histogram experiment. A paired t-test was conducted which compared the best result for each column to the other algorithms in the column. A "-" signifies the top result or tie for top result.

function from the large margin matrix mixture algorithm were similar to a sigmoid function which indicates that useful saturation and threshold effects where uncovered by the method. Figure 8.5 shows examples of training images before and after they have been transformed by the learned function.

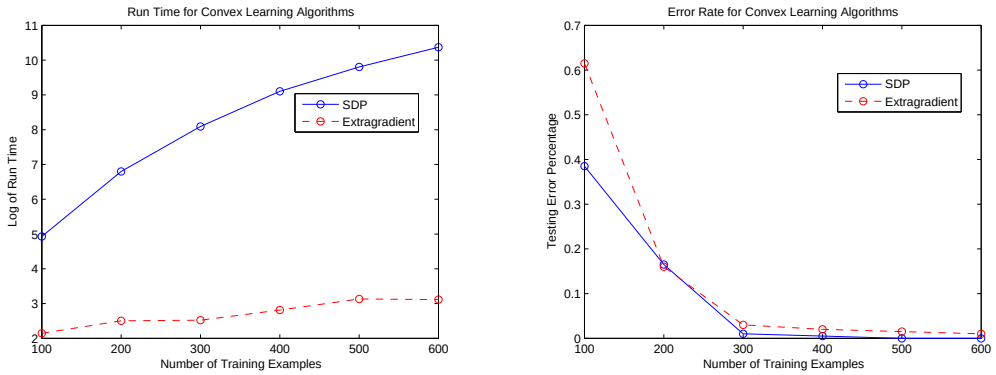
8.6 Empiricial Running Time Comparison for the Extragradient Algorithm

In this experiment, the empirical scaling properties of the extragradient algorithm is compared to solving the semidefinite program with SeDuMi using Yalmip. The transformation learning problem from the synthetic experiment was used with training set sizes of 100, 200, 300, 400, 500 and 600 training examples. At 700 training examples SeDuMi ran out of memory when running a 32 bit version of Matlab which has the limitation of only being able to allocate about 2 GB of memory. For all training regimes, 100 data points were used for cross validation and 100 for testing. The results can be seen in Figures 8.6(a). The empirical scaling of the semidefinite program is $\mathbf{O}(n^{3.06})$ and only $\mathbf{O}(n^{0.7})$ for the extragradient

Algorithm	**Error**	**P Value**
Linear	9.49	0.1170
Mix. Kernels	**8.00**	-
Mix. Trans.	9.66	0.0494
Matrix Mix. Trans.	8.51	0.3586
Trans. Align. Mix	9.71	0.0373

Table 8.7: Percentage error rates for gender classification and P-values for paired t-test. The mixture of transformations (Mix. Trans.), the matrix mixture of transformations (Matrix Mix. Trans.) and the alignment method (Trans. Align. Mix) were compared to a linear SVM (Linear), and a mixture of kernels (Mix. Kernels) to classify gender images.

Figure 8.5: Original and transformed gender images. The top row depicts original images; the second row shows images transformed by learned monotonic functions.



(a) Time comparison (b) Error rate comparison

Figure 8.6: SDP compared to Extragradient to learn matrix mixture of transformation classifiers. Figure (a) shows a comparison of running times for the two algorithms. Figure (b) shows a comparison of error rate, note that this is percentage error.

algorithm. There was a small loss in accuracy for smaller datasets as seen in Figure 8.6(b) which is more than compensated for by the favorable scaling properties on large datasets.

8.7 Empirical Running Time Comparison for Alignment Algorithms

We now compare the empirical running time of the Matrix Mixture of Transformations, Norm Constrained Transformation Learning via Alignment, and Mixing Weight Constrained Transformation Learning via Alignment. The synthetic experiment was varied over training sizes for learning the square root transformation. At 800 training points the Matrix Mixture of Transformations runs out of memory using a 32 bit version of Matlab. Results for the experiments are found in Figure 8.7. The running time for the alignment based algorithms include in addition to the alignment SDP, the running time for a simple dual SVM implemented with quadratic programming and Mosek.

Previous experiments show that mixing weight constrained alignments is a viable alternative to the large margin matrix mixture of transformations in terms of accuracy. The empirical scaling results show that it could be considered as a first algorithm to try be-

Figure 8.7: Empirical scaling results for Large Margin Matrix Mixture of Transformation Learning (LMTL), Norm Constrained Transformation Learning via Alignment (NC-TLA) and Mixing Weight Constrained Transformation Learning via Alignment (MW-TLA).

fore running the more computationally intensive large margin transformation learning. The alignment SDP only scales with the number of transformations and not the data points as opposed to other methods which scale with both the data points and the number of transformations.

8.8 Multitask Learning

In these experiments, the multitask setting is explored where multiple related tasks are linked through a single shared transformation. By learning the same transformation for all tasks, the algorithm can leverage information that would not be available if it were to learn each independently. The first experiment will revisit the synthetic problem and the second experiment will focus on the Corel image histogram classification.

The first experiment uses a similar setup as Section 8.2. In the multitask experiment, four different data sets were generated, each with a different hyperplane. The data generated for each task is then mapped by one common transform (a quadratic). This creates a set of four related tasks which each have a separate hyperplane but all use a common transformation.

Both the greedy mixture of transformations and convex matrix mixture of transformations multitask algorithms were applied to this problem and compared to their independent counterparts. Table 8.8 shows the results of the experiments with the bold entries indicating the lower error when comparing the independent and multitask algorithms. Table 8.9 reports the p-values associated with the paired t-test comparing the single task version to the multitask algorithm. In both cases the error rate was reduced by using the multitask to leverage information across tasks. In fact, both the greedy mixture of transformations and convex matrix mixture of transformations multitask methods had the same accuracy on all four subtasks. This indicates that the multitask setting may help to avoid local minima in the greedy optimization.

The second multitask experiment revisits the Corel image histogram classification task from section 8.3. These tasks naturally fit into the multitask framework because each one of the six pairwise classification problems is related. Again, the two multitask algorithms are

compared to their independent counterparts and the results in Table 8.10 show the benefit of learning a common transformation in this setting. The associated p-values for the paired t-tests are reported in 8.11. The greedy multitask algorithm substantially improved on the independent greedy algorithm and the convex multitask algorithm improves the results to the best reported in this article on this dataset.

The multitask algorithms learn transformations that differ from a simple averaging of individually learned transformations as can be seen in Figure 8.8. The averaged transformations are shown in Figure 8.8(a) and the multitask transformations are shown in Figure 8.8(b). The functions learned with a mixture of transformations are plotted in solid blue and the functions learned with a matrix mixture of transformations are plotted in dashed red. Clearly the averaged functions and the multitask functions differ in that the averaged functions peak more quickly.

	w_1	w_2	w_3	w_4	**Avg.**
Mix. Trans.	1.0	1.8	0.7	0.8	1.07
Mix. Trans. Multitask	**0.4**	**0.3**	**0.3**	**0.0**	**0.25**
Matrix Mix. Trans.	0.6	0.5	0.7	0.4	0.55
Matrix Mix. Trans. Multitask	**0.4**	**0.3**	**0.3**	**0.0**	**0.25**

Table 8.8: Percent testing error rates on synthetic dataset in the multitask framework. Each column contains the error rate for a different dataset based on hyperplanes $\{w1, \ldots, w4\}$. Each dataset is linked by a common transformation which is learned individually and jointly. The multitask learning algorithms are compared to their individual counterparts. Bold entries indicate superior results when comparing independent and multitask algorithms.

	w_1	w_2	w_3	w_4	**Avg.**
Mix. Trans. Multitask	0.2598	0.0811	0.3994	0.0368	0.0309
Matrix Mix. Trans. Multitask	0.5554	0.5911	0.3732	0.1679	0.0510

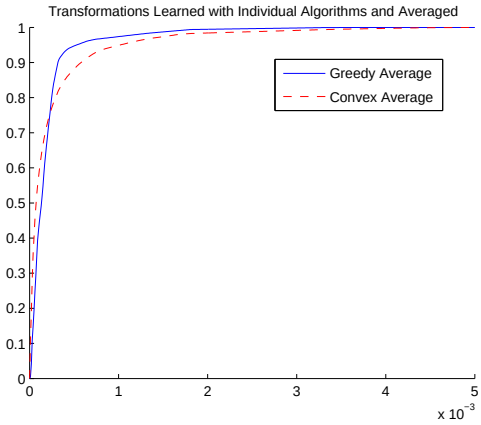
Table 8.9: P-values for the synthetic multitask experiments. A paired t-test was conducted which compared the single task version and the multitask.

	1 vs 2	**1 vs 3**	**1 vs 4**	**2 vs 3**	**2 vs 4**	**3 vs 4**	**Avg.**
Mix. Trans.	3.5	**1.0**	2.0	1.0	4.5	2.5	2.44
Mix. Trans. MT	**2.5**	**1.0**	**1.0**	**0.5**	**1.5**	**1.5**	**1.34**
Mat. Mix. Trans.	2.5	**1.0**	0.5	**1.0**	**2.0**	1.0	1.34
Mat. Mix. Trans. MT	**1.5**	1.5	**0.0**	**1.0**	**2.0**	**0.5**	**1.09**

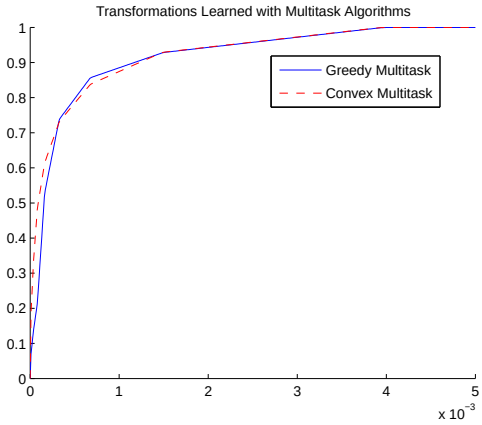
Table 8.10: Percent testing error rates on Corel dataset in the multitask framework. Each column reports one of the six pairwise classification problems from the Corel dataset. The multitask learning algorithms are compared to their individual counterparts. Bold entries indicate superior results when comparing independent and multitask algorithms.

	1 vs 2	**1 vs 3**	**1 vs 4**	**2 vs 3**	**2 vs 4**	**3 vs 4**	**Avg.**
Mix. Trans.	0.1679	-	0.6783	-	0.1679	0.1039	0.1341
Mix. Trans. MT	-	-	-	-	-	-	-
Mat. Mix. Trans.	0.1679	-	0.3434	-	-	0.3434	0.3938
Mat. Mix. Trans. MT	-	0.3434	-	-	-	-	-

Table 8.11: P-values for the multitask Corel image histogram experiments. A paired t-test was conducted which compared the single task to the multitask algorithms. A "-" signifies the top result or tie for top result.



a) Averaged Transformations



b) Multitask Transformations

Figure 8.8: Learned multitask transformations for the Corel image histogram dataset. The average functions for all six tasks learned individually is shown in (a). The functions learned with the multitask algorithms is shown in (b). The greedy mixture of transformations is plotted in solid blue and the convex matrix mixture of transformations is plotted in dashed red.

Chapter 9

Conclusions

9.1 Summary of Contributions

In this dissertation, we have developed a general framework for learning large margin transformations. Chapter one began with a review of kernel learning algorithms and some basics of optimization methods. In chapter two, the large margin transformation framework was introduced to simultaneously learn a maximum margin hyperplane and a mixture of transformations $\Phi(x) = \sum_i m_i \phi_i(x)$ by optimizing an SVM-like cost function. A fast greedy algorithm was proposed to solve this nonconvex optimization problem by iteratively solving a linear program and an SVM optimization. Chapter three looked at convex relaxations of the large margin transformation learning algorithm to find better solutions than the locally optimal ones found with the greedy algorithm. A semidefinite relaxation was derived which led to the matrix mixture of transformations formulation with learned kernels of the form $k(x, x') = \sum_{i,j} M_{i,j} \phi_i(x)^T \phi_i(x')$. Chapter four investigated a faster algorithm to solve the large margin matrix mixture of transformations problem. The structure of this problem was exploited in the extragradient algorithm which made use of fast projections computed via Dykstra's algorithm. Chapter five investigated a multitask setting where information was shared between related tasks in the form of a single globally learned transformation. Chapter six looked at a specific example of transformation learning which built monotonic transformations out of a base set of truncated ramp functions. Chapter seven looked at maximizing the kernel alignment. A semidefinite program was derived to align

the matrix mixture of transformations with the idealized label matrix. Additionally, sparse transformation alignment algorithms were shown to have theoretically guaranteed approximations. Chapter eight demonstrated the efficacy of the proposed algorithms on a diverse set of problems. Finally, this dissertation concludes with the current discussion and some possible future directions.

9.2 Future Directions

There are three clear directions that could easily be explored in the future. The first is to look at other families of learned transformations such as image filters. Second, the fast and flexible extragradient algorithm opens up a range of new transformation learning models such as class specific or even instance specific transformations. A final direction would be to explore tighter convex relaxations.

Image filters are a rich class of transformations to explore. A collection of linear and nonlinear image filters such as edge detectors, blurring filters, sharpening filters, etc. could be used to create a composite image more amenable to classification. Another possibility would be to learn one single linear filter with a linear combination of simple basis filters. For example, this linear filter could find an optimal blur filter for classifying digits data that would incorporate some invariance to translation and rotation. A simple 3X3 linear image filter could be built as linear combination of the following basis filter matrices:

$$F = \left\{ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\} \quad (9.1)$$

There are also some algorithmic extensions that are enabled by the flexible extragradient algorithm. A possible direction could be class specific transformations in a multiclass setting. For example, when classifying hand written digits, each class of digits might have its own learned filter. This leads to the following optimization:

$$\begin{aligned} \min_{\vec{w}_{1,\ldots,R},\vec{\xi},b,\vec{m}_{1,\ldots,R}} \quad & \sum_{r=1}^{R} \|\vec{w}_r\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & \left\langle \vec{w}_{y_i}, \sum_{j=1}^{J} m_{j,y_i}\phi_j(\vec{x_i}) \right\rangle - \left\langle \vec{w}_r, \sum_{j=1}^{J} m_{j,r}\phi_j(\vec{x_i}) \right\rangle \geq 1-\xi_i,\ \forall i, r \neq y_i \\ & \xi_i \geq 0, m_{j,r} \geq 0, \sum_j m_{j,r} \leq 1,\ \forall i,j,r \end{aligned} \tag{9.2}$$

where $\vec{w}_r$ is the hyperplane for the r'th class and $\vec{m}_r$ is the vector of mixing weights for the r'th class with $m_{j,r}$ the j'th component of the mixing weight. In this formulation, the correct classification score $\left\langle \vec{w}_{y_i}, \sum_{j=1}^{J} m_{j,y_i}\phi_j(\vec{x_i}) \right\rangle$ for a training pair (x_i, y_i) must score higher than the score of an incorrect labeling $\left\langle \vec{w}_r, \sum_{j=1}^{J} m_{j,r}\phi_j(\vec{x_i}) \right\rangle$ otherwise a penalty is incurred based on the slack variable. This is a similar setting to other multiclass SVM algorithms [CSC+01] with the addition of class specific transformations. Predictions are made by finding the highest scoring class:

$$y = \text{argmax}_r \left\langle \vec{w}_r, \sum_{j=1}^{J} m_{j,r}\phi_j(\vec{x}) \right\rangle. \tag{9.3}$$

Convex relaxations leading to a matrix mixture of transformations can be also derived, and this relaxed optimization can be solved efficiently with the extragradient algorithm.

Another model extension that might yield interesting results is instance specific transformations. In this setting, there is a separate transformation associated with each training point. This could be an overly rich class of transformations leading to over fitting, so it would be important to use only very simple transformations. Task specific transformation learning can be phrased as the following optimization and again, convex relaxations can be derived and implemented with the extragradient algorithm.

$$\begin{aligned} \min_{\vec{w},\vec{\xi},b,\vec{m}_{1,\ldots,N}} \quad & \|\vec{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i\left(\left\langle \vec{w}, \sum_{j=1}^{J} m_{j,i}\phi_j(\vec{x_i}) \right\rangle + b\right) \geq 1-\xi_i,\ \forall i \\ & \vec{\xi} \geq \vec{0}, \vec{m}_i \geq \vec{0}, \vec{m}_i^T\vec{1} \leq 1,\ \forall i \end{aligned} \tag{9.4}$$

In order to make predictions, a new transformation for each test point must be chosen to maximize the classification score:

$$\hat{m} = \text{argmax}_m \left| \left\langle \vec{w}, \sum_{j=1}^{J} m_j \phi_j(\vec{x}) \right\rangle + b \right| \tag{9.5}$$

$$y = \text{sign} \left(\left\langle \vec{w}, \sum_{j=1}^{J} \hat{m}_j \phi_j(\vec{x}) \right\rangle + b \right).$$

A simple case for instance specific transformation learning would be to learn the best "jittered" image for image classification. The jittered kernel was introduced to try to capture translation invariance in image classification [DS02]. It is defined by finding the minimum pairwise distance in the kernel space between small translations (or jitters) of image data:

$$k(A, A') = \min_{i,j} k(F_i * A, F_i * A') + k(F_j * A, F_j * A') - 2k(F_i * A, F_j * A') \tag{9.6}$$

where A and A' are images, F_i is the i'th simple basis filter from 9.1 and $*$ is the convolution operator. This method is used in a preprocessing step and suffers from two drawbacks. The first is that it does not define a valid kernel; a single image may be translated differently depending on which image is it paired with thus violating the triangle inequality. Another problem is that minimizing pairwise distances does not necessarily imply that classification accuracy is improved. By choosing the best jitter per image with the maximum margin transformation learning algorithm, both of these problems are overcome. Minimizing loss while maximizing margin is a more theoretically sound way to improve prediction accuracy than locally minimizing distance between points, and by defining a single global jitter per point, this method defines valid kernels.

A final possible avenue of exploration would be to look at tighter convex relaxations. In [Las00], a sequence of tighter relaxations involving larger SDPs of increasing complexity was derived and may be useful for large margin transformation learning. The first and simplest relaxation in this sequence has been the method used throughout this dissertation. Each subsequent relaxation introduces higher order monomials leading to a tighter relaxation and

more computation. The relaxed variables take the form of:

$$M = \begin{bmatrix} \vec{m}(0) \\ \vdots \\ \vec{m}(D) \end{bmatrix} \begin{bmatrix} \vec{m}(0)^T & \cdots & \vec{m}(D)^T \end{bmatrix}, M \succeq 0, \tag{9.7}$$

where $\vec{m}(d)$ is the vector of all d order monomials e.g. $m_1^2 m_3$, a third order monomial, is a component of $\vec{m}(3)$ which involves the first and third dimensions of the vector $\vec{m}$. The M matrix must be positive semidefinite and each inequality constraint is replaced with a semidefinite constraint which leads to a large SDP with multiple blocks. Solving this with a standard interior point solver would be impractical for any reasonably sized problem, but the extragradient algorithm may be able to find solutions in a reasonable amount of time by using fast projections via Dykstra's algorithm.

Bibliography

[AHMP06] A. Argyriou, R. Hauser, C. Micchelli, and M. Pontil. A DC-programming algorithm for kernel selection. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 41–48, New York, NY, USA, 2006. ACM.

[AMP05] A. Argyriou, C. Micchelli, and M. Pontil. Learning convex combinations of continuously parameterized basic kernels. In *Learning Theory, 18th Annual Conference on Learning Theory, COLT 2005*, volume 4005 of *Lecture Notes in Computer Science*. Springer, 2005.

[AZ05] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *J. Mach. Learn. Res.*, 6:1817–1853, 2005.

[Bau96] H. Bauschke. On projection algorithms for solving convex feasibility problems. *Journal of Mathematical Analysis and Applications*, 202:150–159, 1996.

[BC64] E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society, Series B*, 26:211–246, 1964.

[BC01] H. Bauschke and L. Combettes. A weak-to-strong convergence principle for Feje-monotone methods in Hilbert spaces. *Mathematics of Operations Research*, 26(2):248–264, 2001.

[BC04] T. De Bie and N. Cristianini. Convex methods for transduction. In *Advances in Neural Information Processing Systems 16*, pages 73–80. MIT Press, 2004.

[BGL+00] M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. S. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene ex-

pression data by using support vector machines. In *Proceedings of the National Academy of Science*, pages 262–267, 2000.

[BH03] O. Bousquet and D. Herrmann. On the complexity of learning the kernel matrix. In *Advances in Neural Information Processing 15*, pages 415–422, 2003.

[Bha43] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math Soc.*, 1943.

[BLJ04] F. Bach, G. Lanckriet, and M Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 6, New York, NY, USA, 2004. ACM.

[Bru55] H. D. Brunk. Maximum likelihood estimates of monotone parameters. *Annals of Mathematical Statistics*, 26:607–616, 1955.

[CHM04] C. Cortes, P. Haffner, and M. Mohri. Rational kernels: Theory and algorithms. *J. Mach. Learn. Res.*, 5:1035–1062, 2004.

[CHV99] O. Chapelle, P. Haffner, and V.N. Vapnik. Support vector machines for histogram-based classification. *Neural Networks, IEEE Transactions on*, 10:1055–1064, 1999.

[CKS02] K. Crammer, J. Keshet, and Y. Singer. Kernel design using boosting. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*, pages 537–544. MIT Press, 2002.

[CMR08] C. Cortes, M. Mohri, and A. Rostamizadeh. Learning sequence kernels. In *Proceedings of IEEE International Workshop on Machine Learning for Signal Processing*, 2008.

[CR08] O. Chapelle and A. Rakotomamonjy. Second order optimization of kernel parameters. In *NIPS Workshop on Automatic Selection of Optimal Kernel*, 2008.

[CSC+01] K. Crammer, Y. Singer, N. Cristianini, J. Shawe-taylor, and B. Williamson. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:2001, 2001.

[CSTK02] N. Cristianini, J. Shawe-Taylor, and J. Kandola. On kernel target alignment, 2002.

[CSZ06] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[CV95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[CVL07] A. B. Chan, N. Vasconcelos, and G. R. G. Lanckriet. Direct convex relaxations of sparse SVM. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 145–153, New York, NY, USA, 2007. ACM.

[DH94] F. Deutsch and H. Hundal. The rate of convergence of Dykstra's cyclic projections algorithm: The polyhedral case. *Numerical Functional Analysis*, 15:537–565, 1994.

[DS02] D. Decoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190, 2002.

[Dur02] C. Durot. Sharp asymptotics for isotonic regression. *Probability theory and related fields*, 122:222–240, 2002.

[Dyk83] R. L. Dykstra. An algorithm for restricted least squares regression. *J. Amer. Stat.*, 78:839–842, 1983.

[EP04] T. Evgeniou and M. Pontil. Regularized multi–task learning. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 109–117. ACM, 2004.

[FL01] U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *J. Algorithms*, 41(2):174–211, 2001.

[GVG08] R. Ganti, N. Vasiloglou, and A. Gray. Hyperkernel based density estimation. In *NIPS Workshop on Automatic Selection of Optimal Kernel*, 2008.

[GW95] M.X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[HB05] M. Hein and O. Bousquet. Hilbertian metrics and positive definite kernels on probability measures. In *Proceedings of Artificial Intelligence and Statistics*, 2005.

[HJ08a] A. Howard and T. Jebara. Large margin transformation learning. *JMLR Accepted*, 2008.

[HJ08b] A. Howard and T. Jebara. Learning monotonic transformations for classification. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[HL02] B.S. He and L. Z. Liao. Improvements of some projection methods for monotone nonlinear variational inequalities. *JOTA*, 112:111–128, 2002.

[HPW73] D. L. Hanson, G. Pledger, and F. T. Wright. On consistency in monotonic regression. *Annals of Statistics*, 1:401–421, 1973.

[HYZ02] Q. Han, Y. Ye, and J. Zhang. An improved rounding method and semidefinite relaxation for graph partition. *Mathematical Programming*, 92(3):509–535, 2002.

[HZ02] E. Halperin and U. Zwick. A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. *Random Struct. Algorithms*, 20(3):382–402, 2002.

[Jeb04] T. Jebara. Multi-task feature and kernel selection for SVMs. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 55, New York, NY, USA, 2004. ACM.

[JKH04] T. Jebara, R. Kondor, and A. Howard. Probability product kernels. *Journal of Machine Learning Research*, 5:819–844, 2004.

[Joa99] T. Joachims. Transductive inference for text classification using support vector machines. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 200–209, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[KBLS08] M. Kloft, U. Brefeld, P. Laskov, and S. Sonnenburg. Non-sparse multiple kernel learning. In *NIPS Workshop on Automatic Selection of Optimal Kernel*, 2008.

[Kho89] E. Khobotov. Modification of the extra-gradient method for solving variational inequalities and certain optimization problems. *USSR Comput. Math. Math. Phys.*, 27(5):120–127, 1989.

[KJ07] R. Kondor and T. Jebara. Gaussian and Wishart hyperkernels. In B. Scholkopf, J.C. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press. To appear.

[KL02] R. I. Kondor and J. D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, pages 315–322, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[Kom88] H. Komiya. Elementary proof for Sion's minimax theorem. *Kodai Mathematics Journal*, 11(1):5–7, 1988.

[Kor76] G. M. Korpelevich. The extragradient method for finding saddle points and other problems. *Ekonomika i Matematicheskie Metody*, 12:747–756, 1976.

[KS05] R. Khardon and R. A. Servedio. Maximum margin algorithms with Boolean kernels. *J. Mach. Learn. Res.*, 6:1405–1429, 2005.

[Las00] J.B. Lasserre. Convergent LMI relaxations for nonconvex quadratic programs. In *Proceedings of 39th IEEE Conference on Decision and Control*, 2000.

[LCB+04] G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[LJN06] D. P. Lewis, T. Jebara, and W. Stafford Noble. Nonstationary kernel combination. In William W. Cohen and Andrew Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 553–560. ACM, 2006.

[MY00] B. Moghaddam and M.H. Yang. Sex with support vector machines. In *Advances in Neural Information Processing 13*, pages 960–966, 2000.

[Nes98] Y. Nesterov. Semidefinite relaxation and nonconvex quadratic optimization. *Optimization Methods and Software*, 9:141–160, 1998.

[Neu28] J. Von Neumann. Zur theorie der gesellschaftsspiele. *Math. Annalen*, 100:295–320, 1928.

[OG08] H. Ouyang and A. Gray. Non-sparse multiple kernel learning. In *NIPS Workshop on Automatic Selection of Optimal Kernel*, 2008.

[OSW05] C. S. Ong, A. J. Smola, and R. C. Williamson. Learning the kernel with hyperkernels. *J. Mach. Learn. Res.*, 6:1043–1071, 2005.

[RBCG08] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

[SBD06] N. Srebro and S. Ben-David. Learning bounds for support vector machines with learned kernel. In Gábor Lugosi and Hans-Ulrich Simon, editors, *Learning Theory, 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006, Proceedings*, volume 4005 of *Lecture Notes in Computer Science*. Springer, 2006.

[Sho87] N.Z. Shor. Quadratic optimization problems. *Tekhnicheskaya Kibernetika*, 1:128–139, 1987.

[Shu00] X. Shusheng. Estimation of the convergence rate of Dykstra's cyclic projection algorithm in polyhedral case. *Acta Mathematicae Applicatae Sinica*, 16(2):217–220, 2000.

[Sil98] J. Sill. The capacity of monotonic functions. *Discrete Applied Mathematics, Special Issue on VC Dimension*, 1998.

[Sio58] M. Sion. On general minimax theorems. *Pacific Journal of Mathematics*, 8(1):171–176, 1958.

[SRSS06] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, 2006.

[SS02] B. Schoelkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002.

[SS04] M. J. Silvapulle and P. K. Sen. *Constrained Statistical Inference: Order, Inequality, and Shape Constraints*. Wiley-Interscience, 2004.

[SSWB00] B. Schölkopf, A. J. Smola, R. C. Williams, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

[TAA03] K. Tsuda, S. Akaho, and K. Asai. The *em* algorithm for kernel matrix completion with auxiliary data. *Journal of Machine Learning Research*, 4:67–81, 2003.

[TLJJ05] B. Taskar, S. Lacoste-Julien, and M. I. Jordan. Structured prediction via the extragradient method. In *Advances in Neural Information Processing Systems 18*, 2005.

[Vap98] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[VB96] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIREV: SIAM Review*, 38, 1996.

[WB98] C. K. I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Trans. Pattern Anal. Mach. Intell*, 20(12):1342–1351, 1998.

[WR95] C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In D. S. Touretzky, M. Mozer, and M. E. Hasselmo, editors, *NIPS*, pages 514–520. MIT Press, 1995.

[YZ03] Y. Ye and J. Zhang. Approximation of dense-$\frac{n}{2}$-subgraph and the complement of min-bisection. *Journal of Global Optimization*, 25(1):55–73, 2003.